# Competency Based Learning Material (CBLM)

# Android Mobile Application Development

**Level-4**

## Module: Working with Android Basic

**Code: CBLM - OU-ICT-AMAD-03-L4-V1**

**National Skills Development Authority**
**Prime Minister's Office**
**Government of the People's Republic of Bangladesh**

# Copyright

Approved by __  th Authority Meeting of NSDA Held on --------------

## How to use this Competency Based Learning Material (CBLM)

The module, Working with Android Basic contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information Sheets.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information Sheets** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information Sheet**. **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information Sheet**. Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information Sheets. This is your opportunity to practise the job. You may need to practise the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Module

# Table of Contents

# Module Content

| **Unit of Competency** | **Work with Android Basic** |
|---|---|
| **Unit Code** | **OU-ICT-AMAD-03-L4-V1** |
| **Module Title** | **Working with Android Basic** |
| Module Descriptor | This module covers the knowledge, skills and attitudes required to work with Android Basic<br>It includes the task of deploying android platform, using main building blocks and creating android user interface |
| Nominal Hours | **50** Hours |
| Lerning Outcome | After completing the practice of the module, the trainees will be able to perform the following jobs:<br>1. Install and setup android development environment<br>2. Use main building blocks<br>3. Create android user interface |

**Assessment Criteria**

1. History of Apps Development is explained.
2. Overview of Android Apps development is described.
3. Computer is configured for setting up android application development environment.
4. Android Apps Development Environment in a computer is explained.
5. Main building block of Android apps is used.
6. Activity life cycle is used.
7. UI Widgets are used.
8. View and Layout are created
9. Intents are created.
10. UI with fragments and action bar is designed.
11. A simple activity layout is designed for some basic user operation.

## Learning Outcome 1: Install and setup android development environment

| | |
|---|---|
| Assessment Criteria | 1. History of Apps Development is explained.<br>2. Overview of Android Apps development is described.<br>3. Computer is configured for setting up android application development environment.<br>4. Android Apps Development Environment in a computer is explained. |
| Conditions and Resources | • Actual workplace or training environment<br>• CBLM<br>• Handouts<br>• Job related tools, equipment, and materials<br>• Multimedia Projector<br>• Paper, Pen, Pencil, and Eraser<br>• Internet Facilities<br>• Whiteboard and Marker |
| Contents | 1. History of Apps Development<br>2. Overview of Android Apps development<br>3. Android application development environment.<br>▪ Android studio<br>▪ SDK (Software Development Kit)<br>▪ NDK (Native Development Kit)<br>▪ Emulator<br>4. Android Apps Development Environment |
| Activities/job/Task | 1. Install JDK and Set Environment Variables with following activities:<br>a. Install the latest version of the Java Development Kit (JDK) on your computer.<br>b. Set the JAVA_HOME environment variable to the installation directory of the JDK.<br>c. Update the PATH environment variable to include the bin directory of the JDK.<br>2. Download and Install Android Studio with following<br>3. activities:<br>a. Download the latest version of Android Studio from the official Android Studio Download page.<br>b. Follow the installation instructions for your operating system (Windows, macOS, or Linux).<br>c. Launch Android Studio after installation and complete the initial setup.<br>4. Configure Android SDK with following activities:<br>a. Open Android Studio and access the SDK Manager.<br>b. Download the necessary Android SDK components, |

| | |
|---|---|
| | including the Android SDK Platform and Android Virtual Device (AVD) components.<br>c. Configure the Android SDK location in Android Studio.<br>5. Set Up an Android Virtual Device (AVD) with following activities<br>a. Use the AVD Manager in Android Studio to create a virtual device for testing.<br>b. Choose a target Android version and device configuration.<br>c. Launch the virtual device and test its functionality.<br>6. Connect a Physical Android Device with following activities:<br>a. If you have a physical Android device, enable Developer Options and USB Debugging.<br>b. Connect your device to the computer via USB and ensure that it is recognized by Android Studio.<br>c. Verify that you can deploy and run an app on the physical device.<br>7. Create a New Android Project with following activities:<br>a. Open Android Studio and create a new Android project.<br>b. Choose a project template and configure project settings.<br>c. Write a simple "Hello World" application and run it on the emulator or a physical device.<br>8. Perform Version Control Integration with following activities:<br>a. Explore and experiment with various features of Android Studio, such as code editor, layout editor, debugging tools, and performance analysis tools.<br>b. Create a simple UI using the layout editor and associate it with the application logic<br><br>9. Deploy and Test on Multiple Devices with following activities:<br>a. Test your Android app on multiple virtual devices with different screen sizes and resolutions.<br>b. If possible, test the app on a physical Android device with different specifications.<br>10. Perform Gradle Build Configuration with following activities:<br>a. Understand the Gradle build system used by Android Studio.<br>b. Modify the build Gradle file of your project to include additional dependencies or configurations.<br>c. Sync the project with Gradle and observe the changes |

| | |
|---|---|
| Training Methods | • Blended<br>• Discussion<br>• Presentation<br>• Demonstration<br>• Guided Practice<br>• Individual Practice<br>• Project Work<br>• Problem Solving<br>• Brainstorming |
| Assessment Methods | Assessment methods may include but not limited to<br>• Written Test<br>• Demonstration<br>• Oral Questioning |

## Learning Experience 1: Install and setup android development environment

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials Install and setup android development environment |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Create Kotlin data types" | 2. Read Information sheet 1: Install and setup android development environment<br>3. Answer Self-check 1: Install and setup android development environment<br>4. Check your answer with Answer key 1: Install and setup android development environment |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job/Task Sheet and Specification Sheet<br>**Job Sheet 1.1:** Install JDK and Set Environment Variables with following activities:<br>a. Install the latest version of the Java Development Kit (JDK) on your computer.<br>b. Set the JAVA_HOME environment variable to the installation directory of the JDK.<br>c. Update the PATH environment variable to include the bin directory of the JDK.<br>**Specification Sheet-1.1:** Install JDK and Set Environment Variables with following<br>**Job Sheet 1.2:** Download and Install Android Studio with following activities:<br>a. Download the latest version of Android Studio from the official Android Studio Download page.<br>b. Follow the installation instructions for your operating system (Windows, macOS, or Linux).<br>c. Launch Android Studio after installation and complete the initial setup.<br>**Specification Sheet-1.2:** Download and |

| | |
|---|---|
| | Install Android Studio with following<br>**Job Sheet 1.3:** Configure Android SDK with following activities:<br>a. Open Android Studio and access the SDK Manager.<br>b. Download the necessary Android SDK components,<br>including the Android SDK Platform and Android<br>Virtual Device (AVD) components.<br>c. Configure the Android SDK location in Android Studio.<br>**Specification Sheet-1.3:** Configure Android SDK with following activities<br>**Job Sheet 1.4:** Set Up an Android Virtual Device (AVD) with following<br>activities<br>a. Use the AVD Manager in Android Studio to create a<br>virtual device for testing.<br>b. Choose a target Android version and device<br>configuration.<br>c. Launch the virtual device and test its functionality.<br>**Specification Sheet-1.4:** Set Up an Android Virtual Device (AVD) with following<br>activities<br>**Job Sheet 1.5:** Connect a Physical Android Device with following activities:<br>a. If you have a physical Android device, enable Developer<br>Options and USB Debugging.<br>b. Connect your device to the computer via USB and<br>ensure that it is recognized by Android Studio.<br>c. Verify that you can deploy and run an app on the<br>physical device.<br>**Specification Sheet-1.5:** Connect a Physical Android Device with following |

| | |
|---|---|
| | **Job Sheet 1.6:** Create a New Android Project with following activities:<br>a. Open Android Studio and create a new Android project.<br>b. Choose a project template and configure project settings.<br>c. Write a simple "Hello World" application and run it on<br>the emulator or a physical device.<br>**Specification Sheet-1.6:**<br>**Job Sheet 1.7:** Perform Version Control Integration with following activities:<br>a. Explore and experiment with various features of Android Studio, such as code editor, layout editor, debugging tools, and performance analysis tools.<br>b. Create a simple UI using the layout editor and associate it with the application logic<br>**Specification Sheet-1.7:** Perform Version Control Integration with following<br>**Job Sheet 1.8:** Explore Android Studio Features with following activities:<br>a. Explore and experiment with various features of Android Studio, such as code editor, layout editor, debugging tools, and performance analysis tools.<br>b. Create a simple UI using the layout editor and associate it with the application logic.<br>**Specification Sheet-1.8:** Explore Android Studio Features with following<br>**Job Sheet 1.9:** Deploy and Test on Multiple Devices with following activities:<br>a. Test your Android app on multiple virtual devices with different screen sizes and resolutions.<br>b. If possible, test the app on a physical Android device with different specifications.<br>**Specification Sheet-1.9:** Deploy and Test on Multiple Devices with following<br>**Job Sheet 1.10:** Perform Gradle Build Configuration with following activities: |

| | |
|---|---|
| | a. Understand the Gradle build system used by Android Studio.<br>b. Modify the build Gradle file of your project to include additional dependencies or configurations.<br>c. Sync the project with Gradle and observe the changes<br>**Specification Sheet-1.10:** Perform Gradle Build Configuration with following |

## Information Sheet 1: Install and Setup Android Development Environment

**Learning Objective:** After completion of this information sheet, the learners will be able to explain, define and interpret the following contents

1.1 History of Apps Development
1.2 Overview of Android Apps development
1.3 Android application development environment.
1.4 Android Apps Development Environment

### 1.1 History of Apps Development

Mobile apps were introduced in the 1980s with the release of the first personal digital assistants (PDAs). However, such apps did not evolve far past the most basic and utilitarian of functions (e.g., clocks and calculators) until the 21st century, when smartphones evolved to run larger programs.

The history of app development dates to the early days of computing and mobile devices. It can be divided into several key phases:

a. **Pre-smartphone era (1940s-1990s):** The concept of application software began with the first electronic computers in the 1940s. Early applications were primarily focused on scientific and mathematical calculations. As personal computers gained popularity in the 1980s, software developers started creating various programs for specific tasks, such as word processing, spreadsheets, and graphics.

b. **Birth of mobile apps (1990s-2000s):** With the advent of mobile phones, particularly the first iPhone in 2007, the demand for mobile applications grew rapidly. The first mobile app store, called "AppStore," was introduced by Apple in 2008, which revolutionized the way people downloaded and used apps.

c. **The rise of smartphones and app stores (2010s):** The release of Android OS in 2008, followed by its app store, Google Play, in 2009, further fueled the growth of mobile app development. This decade saw a massive increase in the number of smartphone users and app developers, leading to a vast array of apps catering to various needs and interests.

d. **Mobile-first approach (2010s-present):** As smartphones became the primary device for internet access, businesses and developers shifted their focus to creating mobile-first experiences. This led to the development of responsive web design and progressive web apps (PWAs), which aimed to provide a seamless user experience across different devices.

e. **Emergence of cross-platform development (2010s-present):** To reduce development time and costs, cross-platform frameworks like React Native, Xamarin, and Flutter gained popularity. These tools allowed developers to write code once and deploy it across multiple platforms, including iOS and Android.

f. **Internet of Things (IoT) and Wearables (2010s-present):** The growth of connected devices, such as smartwatches, fitness trackers, and home automation systems, has led to the development of apps specifically designed for these devices. This has expanded the scope of app development beyond traditional smartphones and tablets.

g. **Artificial Intelligence and Machine Learning (2010s-present):** The integration of AI and ML in app development has led to the creation of intelligent apps that can learn from user behavior and provide personalized experiences. Examples include virtual assistants like Siri, Google Assistant, and Amazon Alexa.

h. **Augmented Reality and Virtual Reality (2010s-present):** The rise of AR and VR technologies has opened new opportunities for app developers, allowing them to create immersive experiences

## 1.2 Overview of Android Apps development

Android app development is the process of creating software for Android devices using the Android software development kit (SDK) and a programming language. The Android operating system (OS) is built on Linux and has a layered architecture that allows communication with device components, users, and software applications.

**The typical stages of Android app development are**

- Gathering and analyzing requirements
- Designing the UI/UX
- Developing the front- and back-end
- Integrating APIs and functionalities
- Testing and debugging
- Deploying to the Google Play Store
- Monitoring, updating, and maintaining after launch

Some programming languages that can be used for Android app development include:

**a.** **Java**: A primary language for Android app development
**b.** **Kotlin**: An official language introduced by Google in 2017 that is similar to Java but easier to use. It is integrated into Android Studio and is used by more than 50% of professional Android developers.
**c.** **C# and C++:** Compatible with Android Studio and iOS mobile development software. C# is popular for game development and command-line scripting.
**d.** **Python:** Useful for scripting and rapid prototyping

**Why Java?**
Java is the primary language used for Android app development because of its unique features, advantages, and historical context. Here are some reasons why Java is widely used for Android development:

Early adoption: Android's creator, Andy Rubin, was a Sun Microsystems employee, and Java was the primary language used at Sun Microsystems. This led to Java being chosen as the primary language for Android development from the beginning.

**Platform independence:** Java is a platform-independent language, which means that programs written in Java can run on multiple platforms without modification. This aligns well with Android's goal of providing a platform-agnostic operating system.

**Large community and resources:** Java has a massive community of developers, which means there are numerous resources available for learning and troubleshooting. This is especially important for a new operating system like Android.

**Object-oriented programming:** Java is an object-oriented language, which makes it well-suited for building complex applications with modular, reusable code. Android's architecture is designed around objects and interfaces, making Java a natural fit.

**Android's JVM:** Android uses a customized version of the Java Virtual Machine (JVM), called the Dalvik virtual machine (DVM). This allows Android apps to run on top of the DVM, making it easier to develop and deploy Java-based apps.

**Familiarity for many developers:** Many developers already knew Java or were familiar with the language due to its widespread use in other areas like web development, desktop applications, and enterprise software development.

**Android's SDK:** The Android Software Development Kit (SDK) provides tools and APIs specifically designed for Java development. The SDK includes libraries, frameworks, and other tools that make it easier to build Android apps using Java.

**Kotlin support:** Although Kotlin is now officially supported by Google as a first-class citizen for Android development, Java remains the most widely used language for Android app development.

In summary, Java was chosen as the primary language for Android development due to its platform independence, large community, object-oriented nature, and compatibility with the Dalvik virtual machine. While Kotlin has gained popularity as an alternative to Java for Android development, Java remains a widely used and accepted language for building Android apps.

**Why Kotlin?**
Kotlin is getting high popularity among all level of programmers and it is used for:

- Cross-platform Mobile applications.
- Android Application Development.
- Web Application Development
- Server-Side Applications
- Desktop Application Development
- Data science-based applications
- Kotlin works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.) and it's 100% compatible with Java.

Kotlin is used by many large companies like Google, Netflix, Slack, Uber etc to develop their Android based applications.

The most importantly, there are many companies actively looking for Kotlin developers, especially in the Android development space.

**Kotlin Version?**
At the time of writing this tutorial on Aug 3, 2021, The current Kotlin released version is 1.5.21

**Kotlin Advantages**
Following is some of the advantages of using Kotlin for your application development.

- Easy Language − Kotlin supports object-oriented and functional constructs and very easy to learn. The syntax is pretty much like Java, hence for any Java programmer it is very easy to remember any Kotlin Syntax.

- Very Concise − Kotlin is based on Java Virtual Machine (JVM) and it is a functional language. Thus, it reduce lots of boiler plate code used in other programming languages.

- Runtime and Performance − Kotlin gives a better performance and small runtime for any application.

- Interoperability − Kotlin is mature enough to build an interoperable application in a less complex manner.

- Brand New − Kotlin is a brand-new language that gives developers a fresh start. It is not a replacement of Java, though it is developed over JVM. Kotlin has been accepted as the first official language of Android Application Development. Kotlin can also be defined as - Kotlin = Java + Extra updated new features.

**Kotlin Drawbacks**

Following is some of the disadvantages of using Kotlin.

- Namespace declaration − Kotlin allows developers to declare the functions at the top level. However, whenever the same function is declared in many places of your application, then it is hard to understand which function is being called.

- No Static Declaration − Kotlin does not have usual static handling modifier like Java, which can cause some problem to the conventional Java developer.

## 1.3 Android application development environment.

**Install Android Studio**

Android Studio is the official IDE (Integrated Development Environment) for Android app development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- A flexible Gradle-based build system
- A fast and feature-rich emulator
- GitHub and Code template integration to assist you in developing common app features and importing sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine, and many more.
- Provides GUI tools that simplify the less interesting parts of app development.

- Easy integration with real-time database ‘firebase‘.

Set up Android Studio in just a few clicks. First, check the system requirements. Then download the latest version of Android Studio.

**Windows**

Note: Windows machines with ARM-based CPUs are not currently supported.
Here are the system requirements for Windows:

| Requirement | Minimum | Recommended |
|---|---|---|
| OS | 64-bit Microsoft Windows 8 | Latest 64-bit version of Windows |
| RAM | 8 GB RAM | 16 GB RAM or more |
| CPU | x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor Framework. | Latest Intel Core processor |
| Disk space | 8 GB (IDE and Android SDK and Emulator) | Solid state drive with 16 GB or more |
| Screen resolution | 1280 x 800 | 1920 x 1080 |

**Mac**

Here are the system requirements for Mac:

| Requirement | Minimum | Recommended |
|---|---|---|
| OS | MacOS 10.14 (Mojave) | Latest version of MacOS |
| RAM | 8 GB RAM | 16 GB RAM or more |
| CPU | Apple M1 chip, or 2nd generation Intel Core or newer with support for Hypervisor Framework. | Latest Apple Silicon chip |
| Disk space | 8 GB (IDE and Android SDK and Emulator) | Solid state drive with 16 GB or more |
| Screen resolution | 1280 x 800 | 1920 x 1080 |

**Linux**

**Note:** Linux machines with ARM-based CPUs are not currently supported.

Here are the system requirements for Linux:

| Requirement | Minimum | Recommended |
|---|---|---|

| | | |
|---|---|---|
| OS | Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later. | Latest 64-bit version of Linux |
| RAM | 8 GB RAM | 16 GB RAM or more |
| CPU | x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3. | Latest Intel Core processor |
| Disk space | 8 GB (IDE and Android SDK and Emulator) | Solid state drive with 16 GB or more |
| Screen resolution | 1280 x 800 | 1920 x 1080 |

**Step 1 - System Requirements**

You will be delighted, to know that you can start your Android application development on either of the following operating systems −

- Microsoft® Windows® 10/8/7/Vista/2003 (32 or 64-bit)
- Mac® OS X® 10.8.5 or higher, up to 10.9 (Mavericks)
- GNOME or KDE desktop

Second point is that all the required tools to develop Android applications are open source and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Java Runtime Environment (JRE) 6
- Android Studio

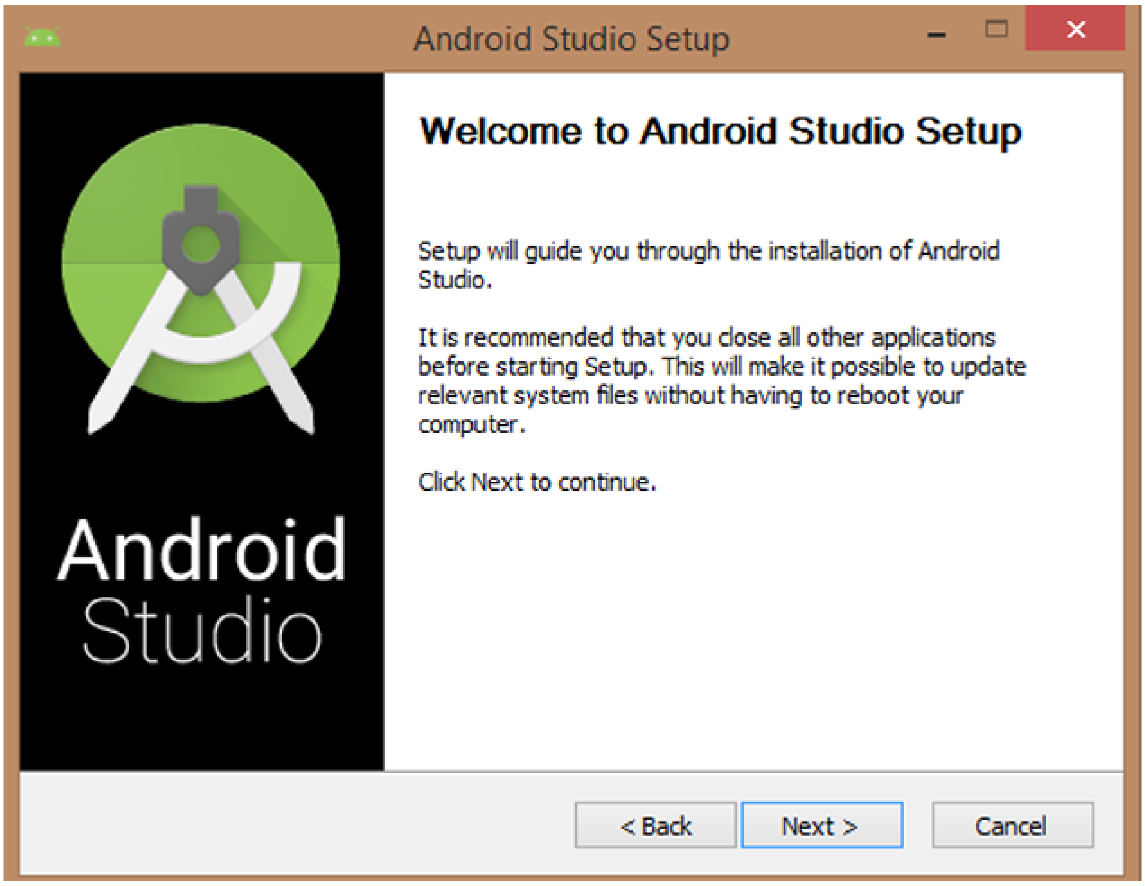
**Step 2 - Setup Android Studio**

**Overview**

Android Studio is the official IDE for android application development. It works based on IntelliJ IDEA, you can download the latest version of android studio from Android Studio 2.2 Download, If you are new to installing Android Studio on windows, you will find a file, which is named as android-studio-bundle-143.3101438 windows.exe. So just download and run on windows machine according to android studio wizard guideline. If you are installing Android Studio on Mac or Linux, you can download the latest version from Android Studio Mac Download, or Android Studio Linux Download, check the instructions provided along with the downloaded file for Mac OS and Linux. This tutorial will consider that you are going to setup your environment on Windows machine having Windows 8.1 operating system.

**Installation**

So, let us launch Android Studio.exe, Make sure before launch Android Studio, Our Machine should require installed Java JDK. To install Java JDK, take a references of Android environment setup



Android Studio Setup

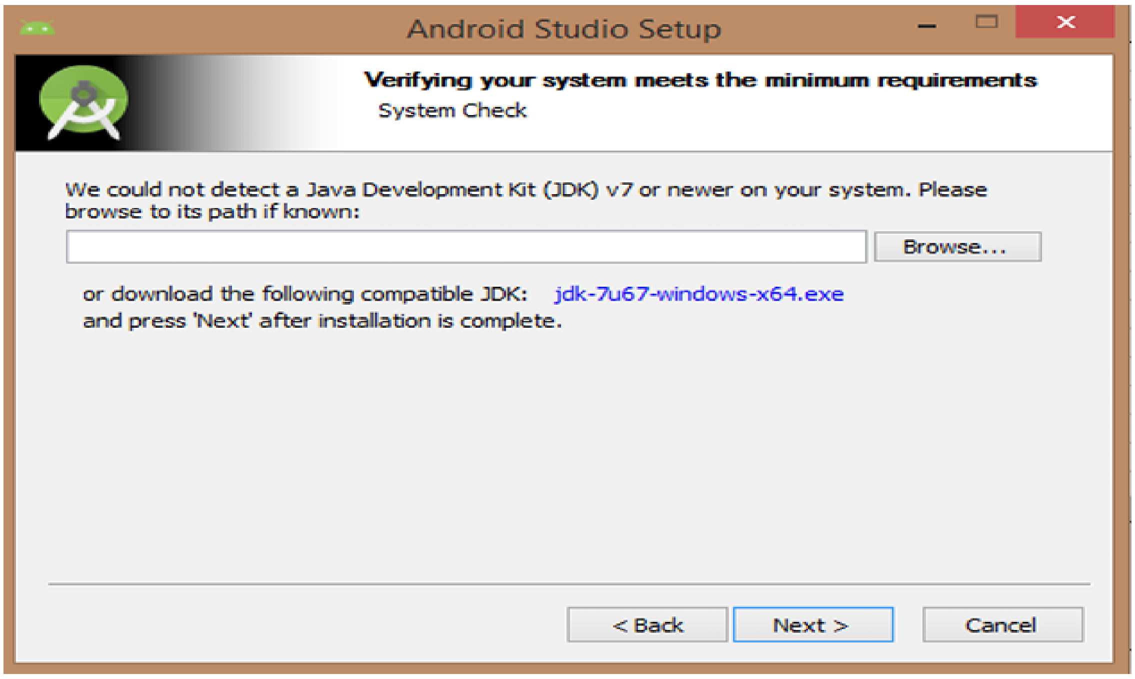
Welcome to Android Studio Setup

Setup will guide you through the installation of Android Studio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Once you launched Android Studio, its time to mention JDK7 path or later version in android studio installer.



Android Studio Setup

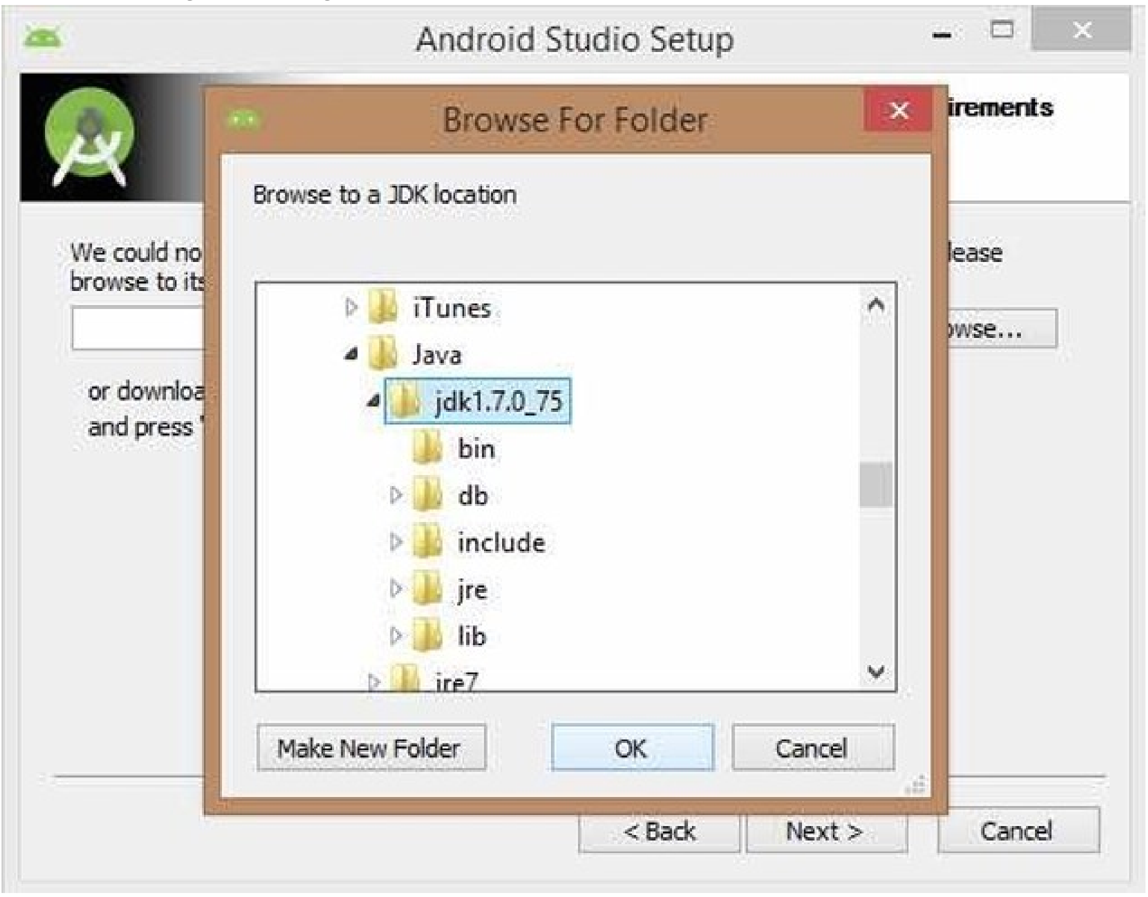
Verifying your system meets the minimum requirements

System Check

We could not detect a Java Development Kit (JDK) v7 or newer on your system. Please browse to its path if known:

or download the following compatible JDK: jdk-7u67-windows-x64.exe and press 'Next' after installation is complete.

Below the image initiating JDK to android SDK



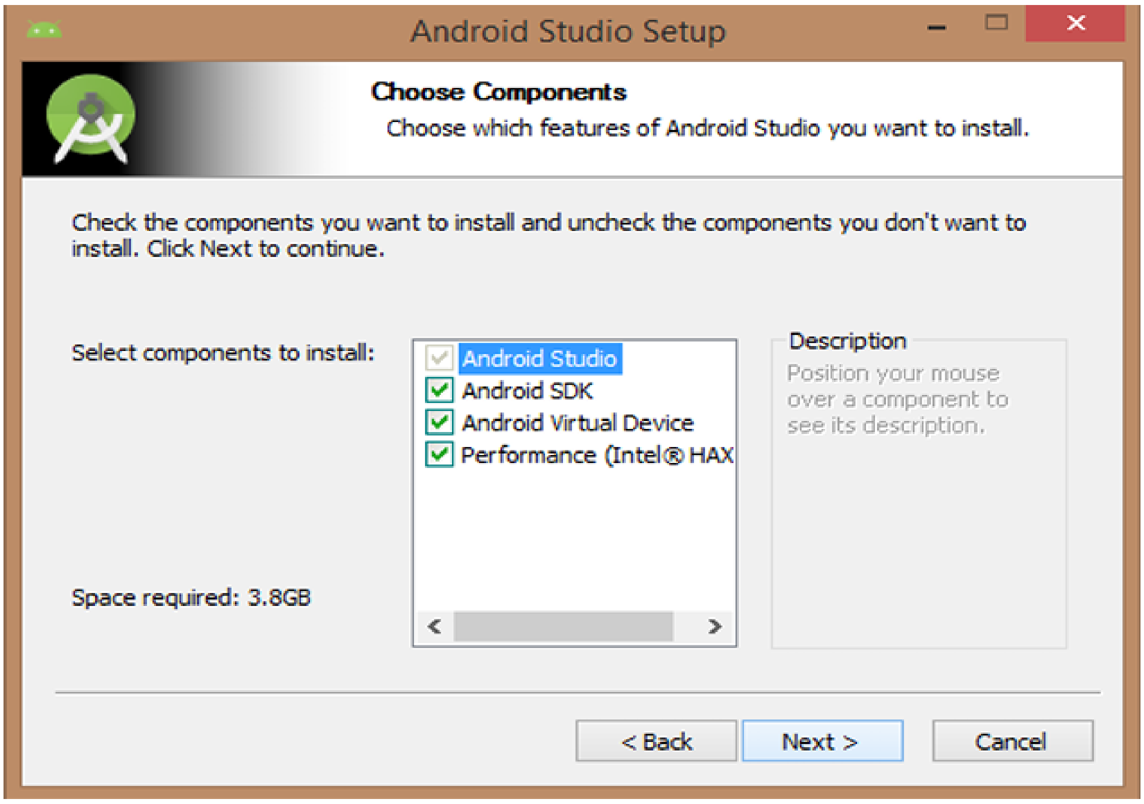
Android Studio Setup

Browse For Folder

Browse to a JDK location

- iTunes
- Java
  - jdk1.7.0_75
    - bin
    - db
    - include
    - jre
    - lib
  - jre7

image has selected Android Studio, Android SDK, Android Virtual Machine and performance(Intel chip).



Android Studio Setup

**Choose Components**

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

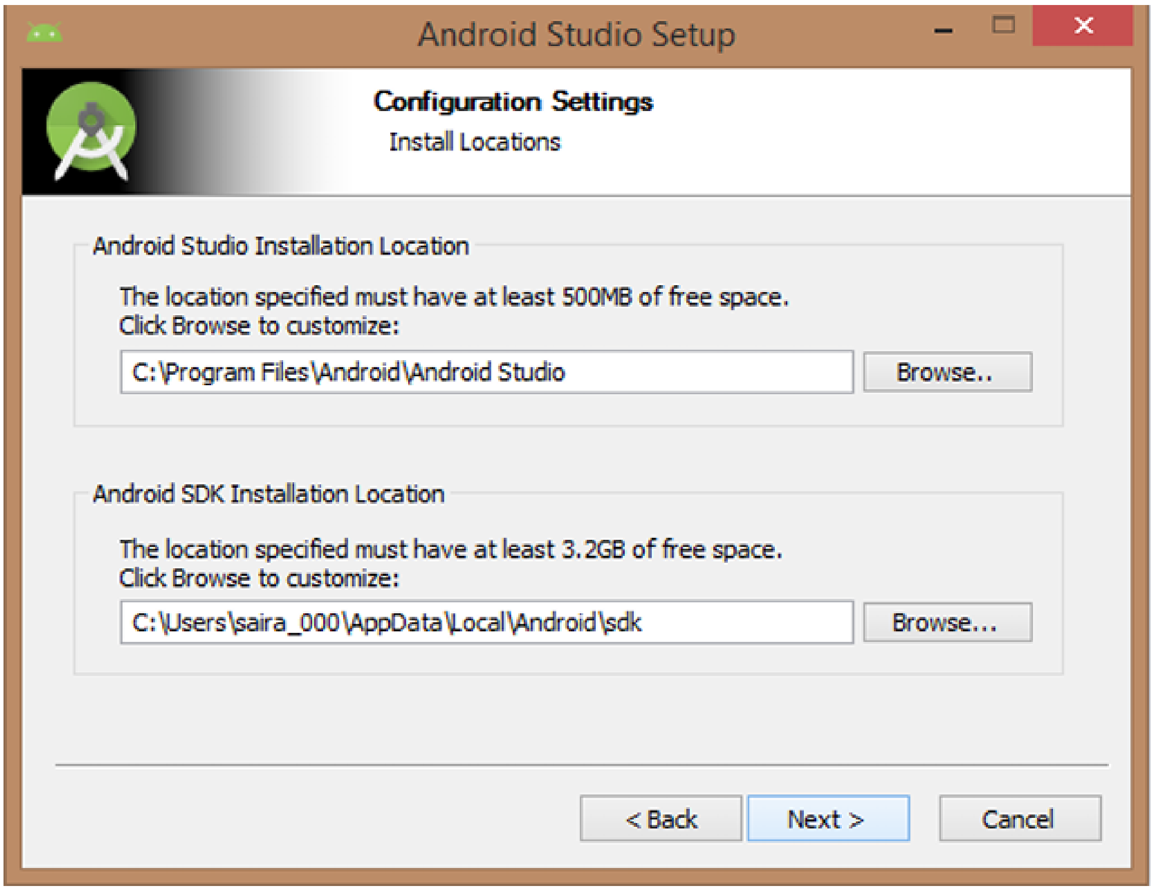
- [✓] Android Studio
- [✓] Android SDK
- [✓] Android Virtual Device
- [✓] Performance (Intel® HAX

Description

Position your mouse over a component to see its description.

Space required: 3.8GB

Need to specify the location of local machine path for Android studio and Android SDK, below the image has taken default location of windows 8.1 x64 bit architecture.



Android Studio Installation Location

The location specified must have at least 500MB of free space.
Click Browse to customize:

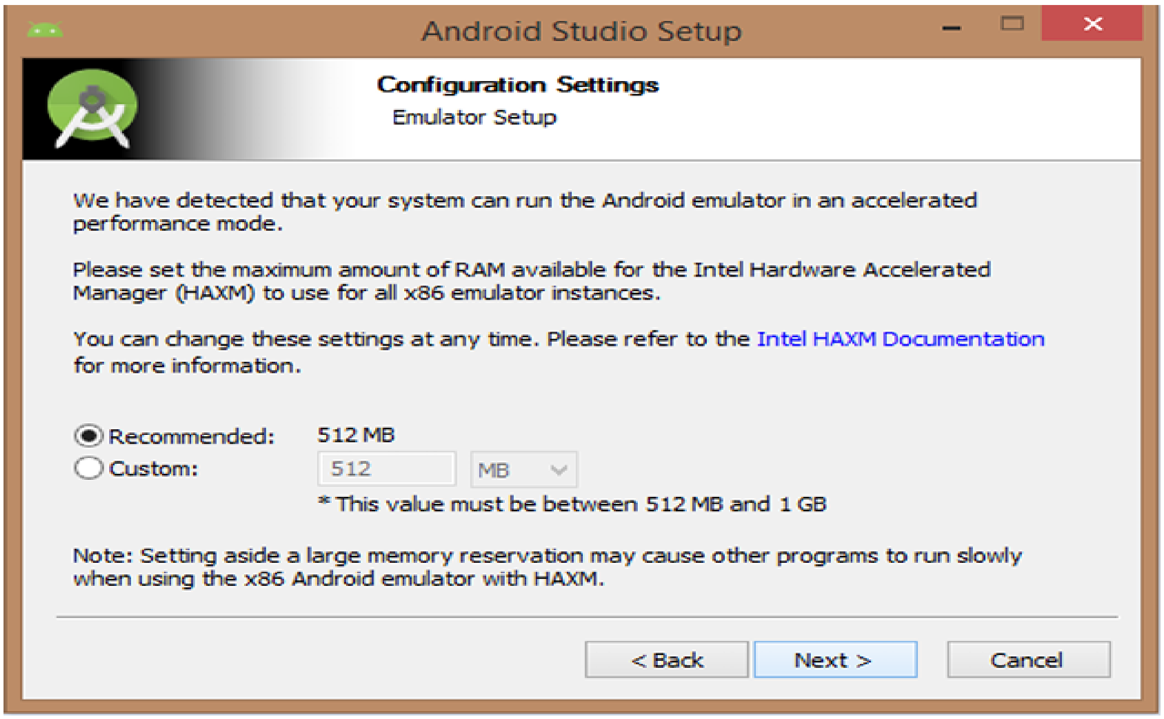
C:\Program Files\Android\Android Studio

Android SDK Installation Location

The location specified must have at least 3.2GB of free space.
Click Browse to customize:

C:\Users\saira_000\AppData\Local\Android\sdk

Need to specify the ram space for Android emulator by default it would take 512MB of local machine RAM.



We have detected that your system can run the Android emulator in an accelerated performance mode.

Please set the maximum amount of RAM available for the Intel Hardware Accelerated Manager (HAXM) to use for all x86 emulator instances.

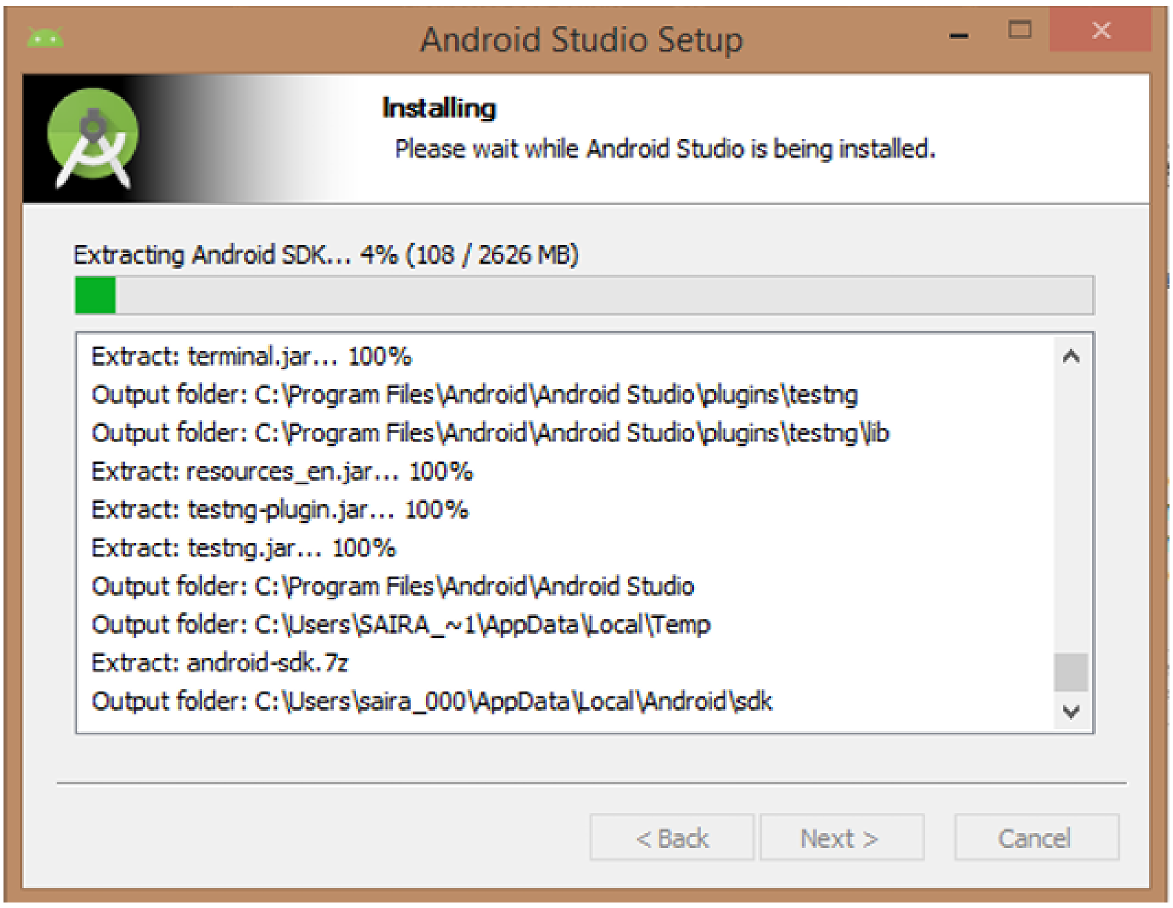
You can change these settings at any time. Please refer to the Intel HAXM Documentation for more information.

● Recommended: 512 MB
○ Custom: 512 MB

* This value must be between 512 MB and 1 GB

Note: Setting aside a large memory reservation may cause other programs to run slowly when using the x86 Android emulator with HAXM.

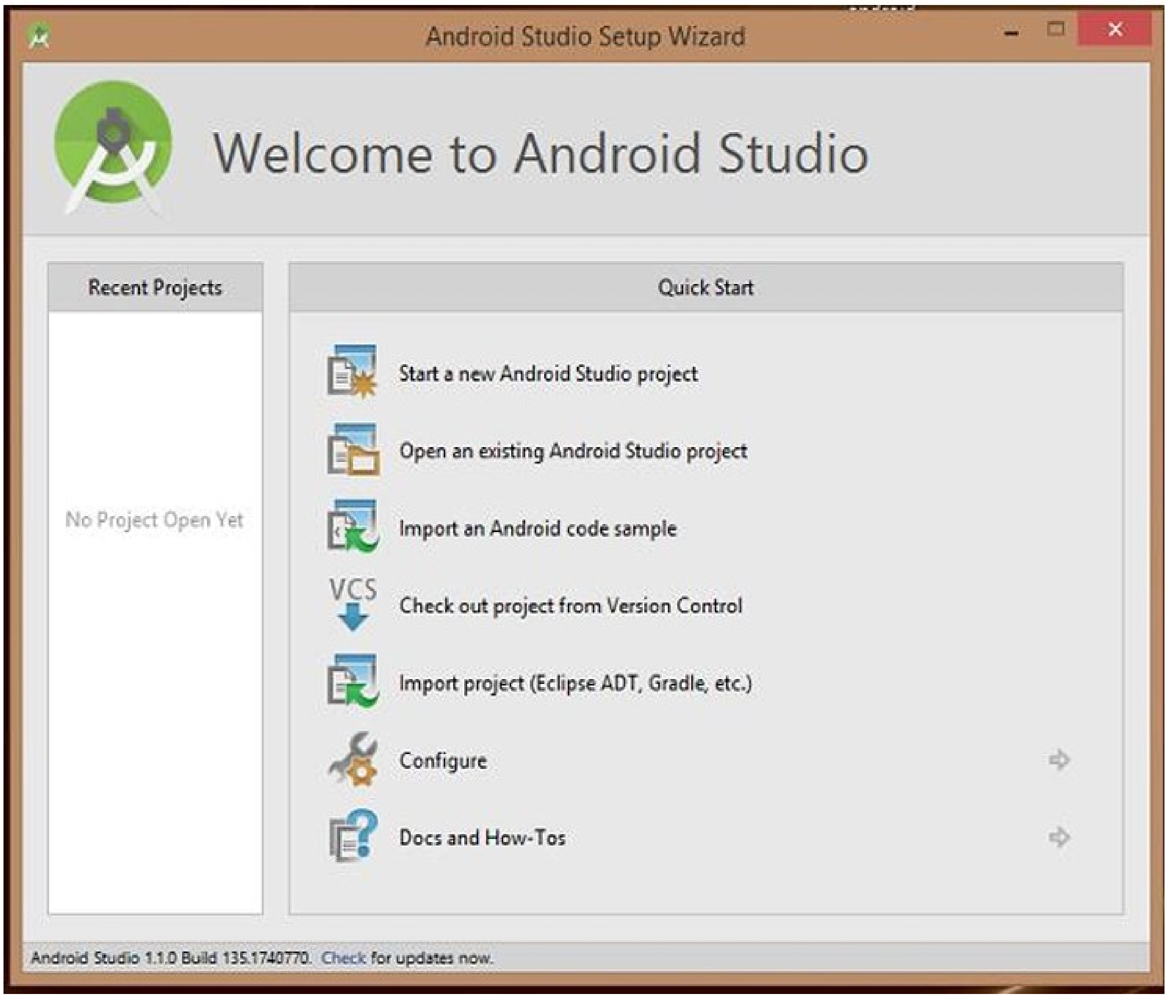
At final stage, it would extract SDK packages into our local machine, it would take a while time to finish the task and would take 2626MB of Hard disk space.



```
Extracting Android SDK... 4% (108 / 2626 MB)
```

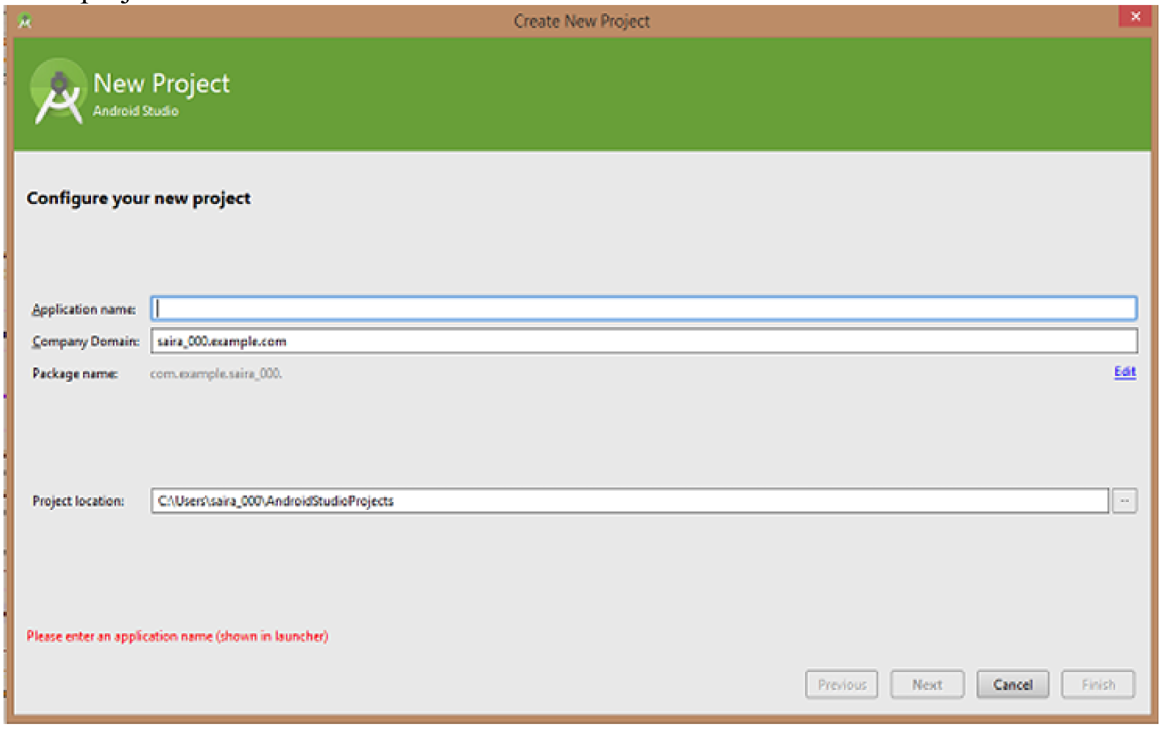
```
Extract: terminal.jar... 100%
Output folder: C:\Program Files\Android\Android Studio\plugins\testng
Output folder: C:\Program Files\Android\Android Studio\plugins\testng\lib
Extract: resources_en.jar... 100%
Extract: testng-plugin.jar... 100%
Extract: testng.jar... 100%
Output folder: C:\Program Files\Android\Android Studio
Output folder: C:\Users\SAIRA_~1\AppData\Local\Temp
Extract: android-sdk.7z
Output folder: C:\Users\saira_000\AppData\Local\Android\sdk
```

After done all above steps perfectly, you must get finish button and it gonna be open android studio project with Welcome to android studio message as shown below



Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos
Android Studio 1.1.0 Build 135.1740770. Check for updates now.

You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.

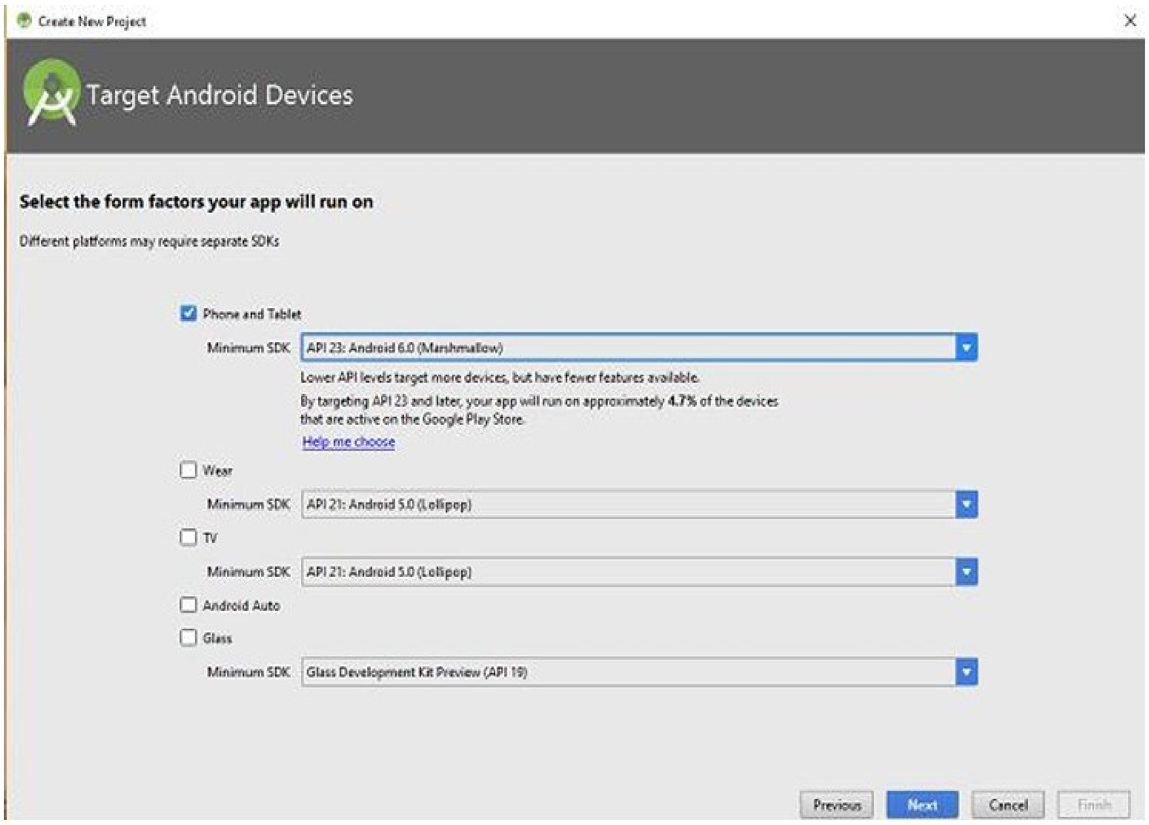


Configure your new project

Application name:
Company Domain: saira_000.example.com
Package name: com.example.saira_000.
Project location: C:\Users\saira_000\AndroidStudioProjects

Please enter an application name (shown in launcher)

After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0 (Mashmallow)



Create New Project

Target Android Devices

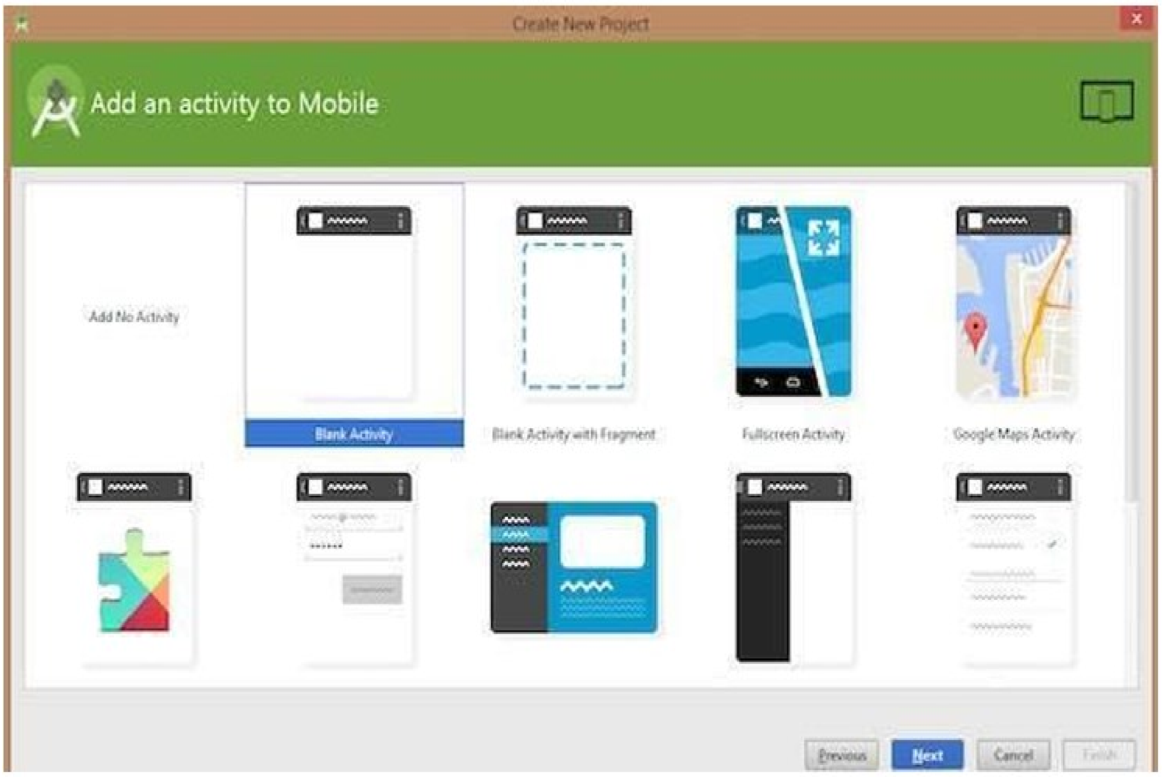
**Select the form factors your app will run on**

Different platforms may require separate SDKs

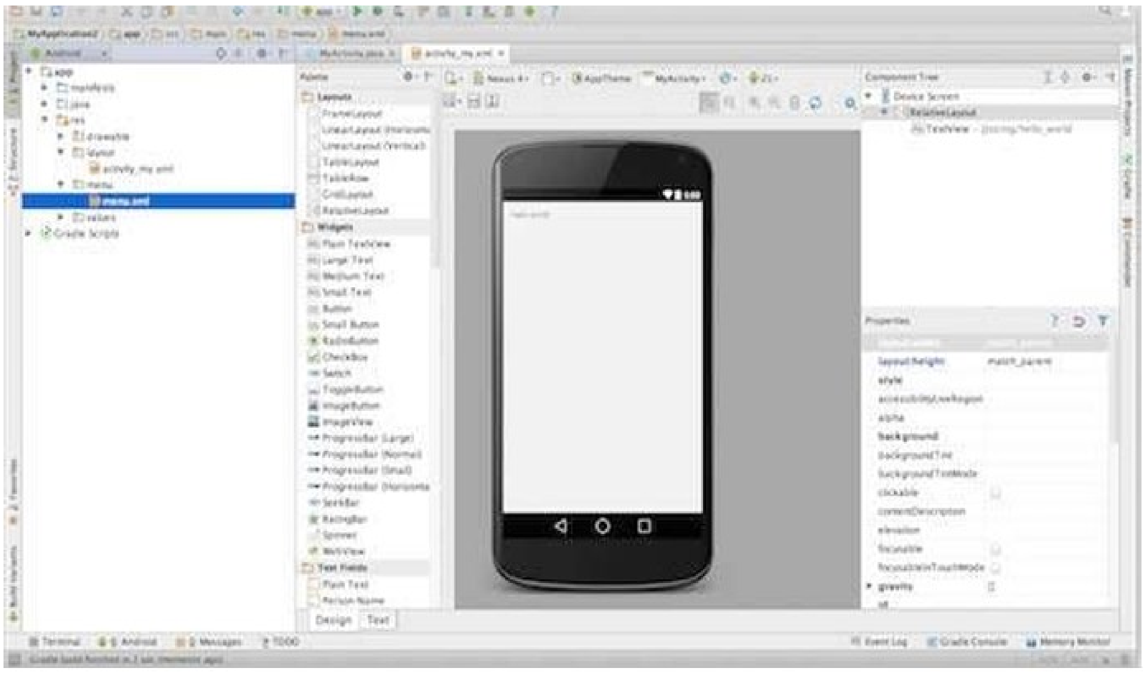
- [x] Phone and Tablet
  - Minimum SDK: API 23: Android 6.0 (Marshmallow)
  - Lower API levels target more devices, but have fewer features available.
  - By targeting API 23 and later, your app will run on approximately **4.7%** of the devices that are active on the Google Play Store.
  - Help me choose
- [ ] Wear
  - Minimum SDK: API 21: Android 5.0 (Lollipop)
- [ ] TV
  - Minimum SDK: API 21: Android 5.0 (Lollipop)
- [ ] Android Auto
- [ ] Glass
  - Minimum SDK: Glass Development Kit Preview (API 19)

Previous | Next | Cancel | Finish

The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications

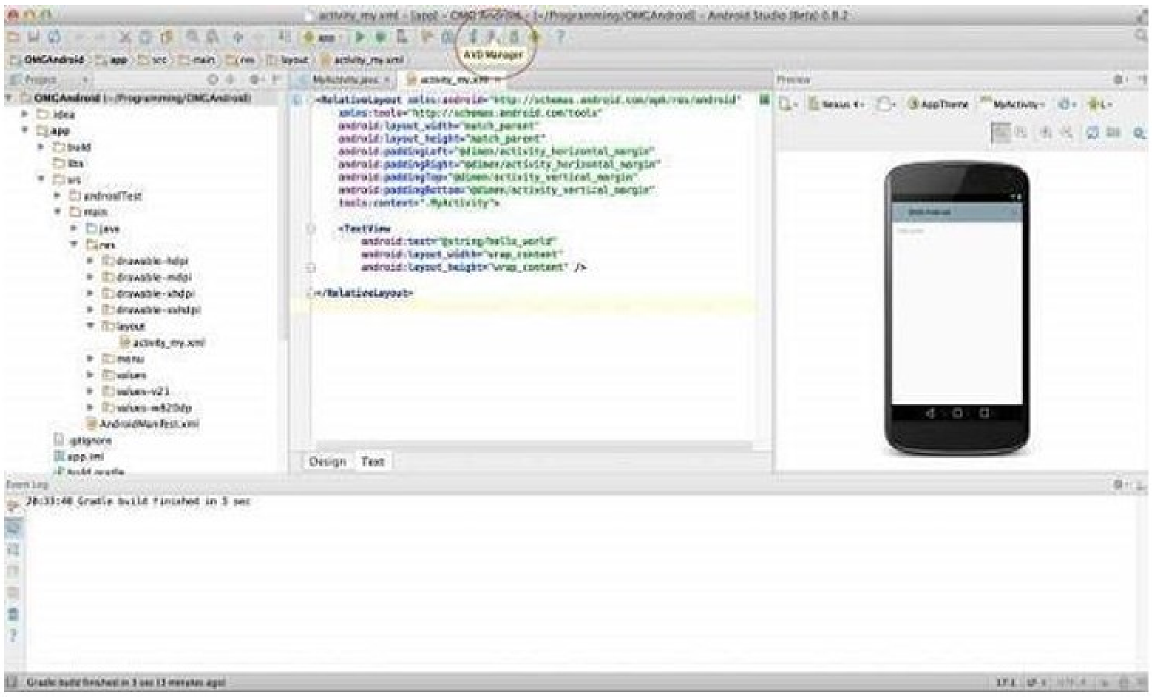


At the final stage it going to be open development tool to write the application code.



### Step 3 - Create Android Virtual Device

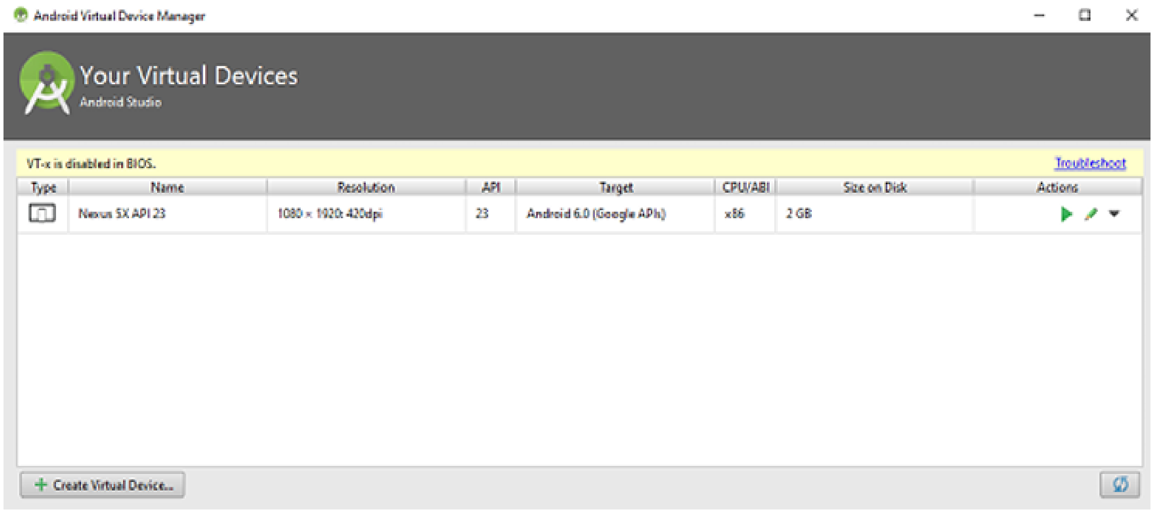
To test your Android applications, you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager Clicking AVD Manager icon as shown below



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

```
20:33:40 Gradle build finished in 3 sec
```

After Click on a virtual device icon, it going to be shown by default virtual devices which are present on your SDK, or else need to create a virtual device by clicking Create new Virtual device button.



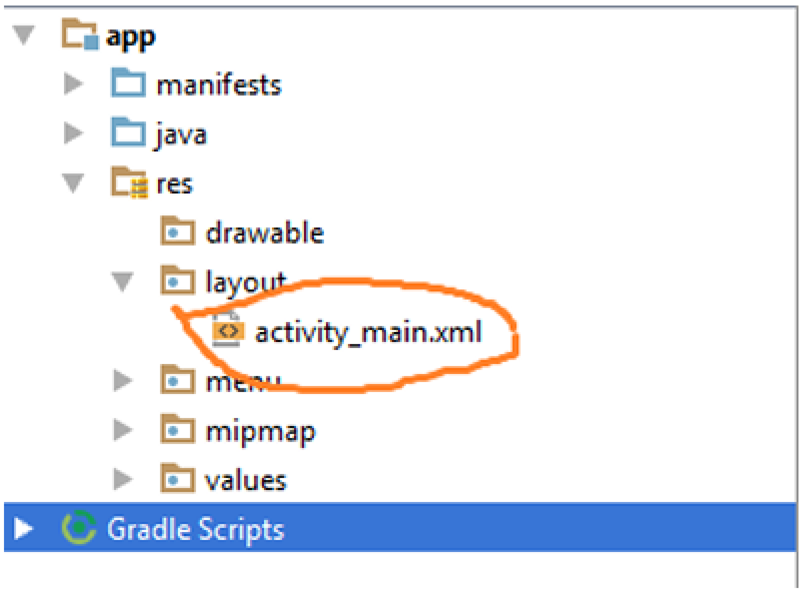
VT-x is disabled in BIOS.

| Type | Name | Resolution | API | Target | CPU/ABI | Size on Disk | Actions |
|---|---|---|---|---|---|---|---|
| | Nexus 5X API 23 | 1080 × 1920: 420dpi | 23 | Android 6.0 (Google APIs) | x86 | 2 GB | |

If your AVD is created successfully it means your environment is ready for Android application development. If you like, you can close this window using top-right cross button. Better you re-start your machine and once you are done with this last step, you are ready to proceed for your first Android example but before that we will see few more important concepts related to Android Application Development.

**Hello Word Example**

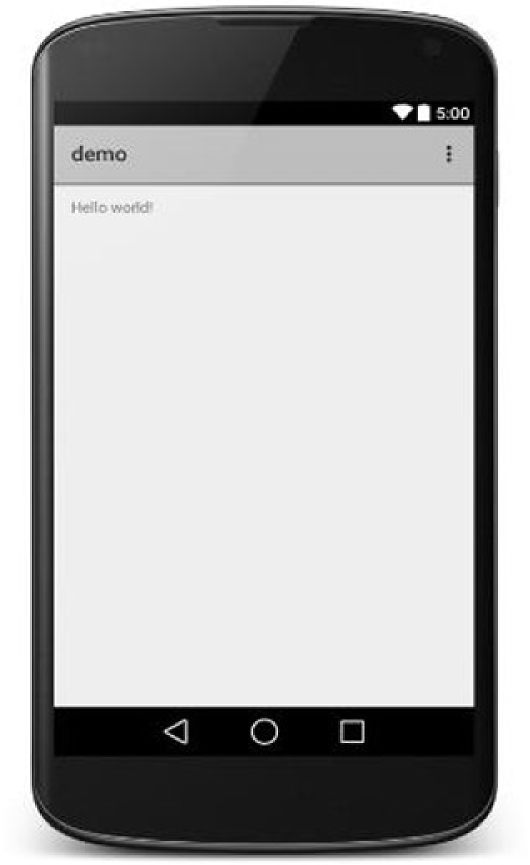
Before Writing a Hello word code, you must know about XML tags. To write hello word code, you should redirect to App>res>layout>Activity_main.xml



To show hello word, we need to call text view with layout ( about text view and layout, you must take references at Relative Layout and Text View ).

```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:paddingBottom="@dimen/activity_vertical_margin"
        tools:context=".MainActivity">
        <TextView android:text="@string/hello_world"
        android:layout_width="550dp"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Need to run the program by clicking Run>Run App or else need to call shift+f10key. Finally, result should be placed at Virtu al devices as shown below



**a. Emulator**

The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device. The emulator offers these advantages:

**Flexibility:** In addition to being able to simulate a variety of devices and Android API levels, the emulator comes with predefined configurations for various Android phone, tablet, Wear OS, and Android TV devices.

**High fidelity:** The emulator provides almost all the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.

**Speed:** Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

In most cases, the emulator is the best option for your testing needs. This page covers the core emulator functionalities and how to get started with it.

Alternatively, you can deploy your app to a physical device. For more information, see Run apps on a hardware device.
Create an Android Virtual Device
Each instance of the Android Emulator uses an Android virtual device (AVD) to specify the Android version and hardware characteristics of the simulated device. To create an AVD, see Create and manage virtual devices.

Each AVD functions as an independent device with its own private storage for user data, SD card, and so on. By default, the emulator stores the user data, SD card data, and cache in a directory specific to that AVD. When you launch the emulator, it loads the user data and SD card data from the AVD directory.

Run your app on the emulator
After you have created an AVD, you can start the Android Emulator and run an app in your project:

In the toolbar, select the AVD that you want to run your app on from the target device menu.

The target device menu
Figure 1. The target device menu.

Click Run. The emulator might take a minute or so to launch for the first time, but subsequent launches use a snapshot and should launch faster. If you experience issues, see the troubleshooting guide.

Once your app is installed on your AVD, you can run it from the device as you would run any app on a device. Any time you want to deploy new changes, you need to click Run or Apply Changes again.

**Create an Android Virtual Device**

Each instance of the Android Emulator uses an Android virtual device (AVD) to specify the Android version and hardware characteristics of the simulated device. To create an AVD, see Create and manage virtual devices.

Each AVD functions as an independent device with its own private storage for user data, SD card, and so on. By default, the emulator stores the user data, SD card data, and cache in a directory specific to that AVD. When you launch the emulator, it loads the user data and SD card data from the AVD directory.

**Run your app on the emulator**

After you have created an AVD, you can start the Android Emulator and run an app in your project:

In the toolbar, select the AVD that you want to run your app on from the target device menu.

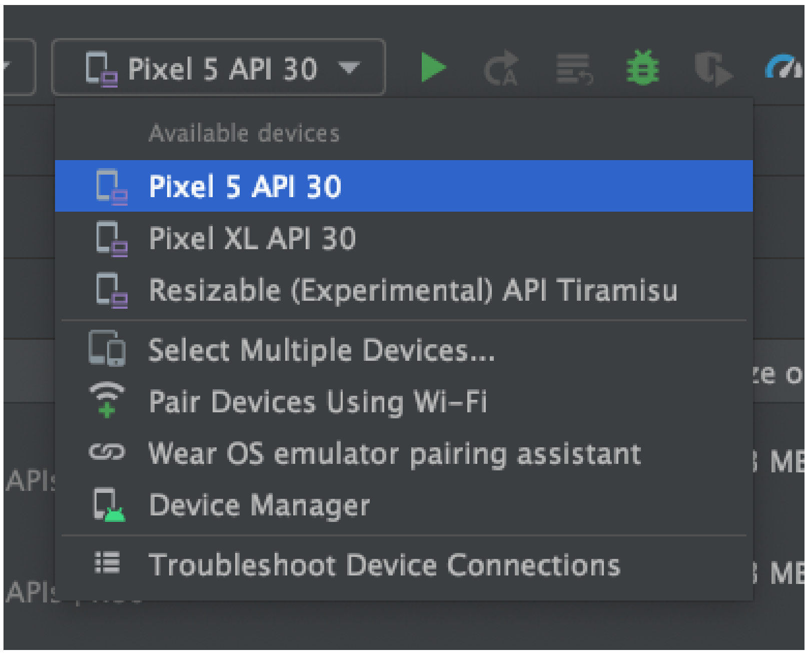

Pixel 5 API 30
Available devices
Pixel 5 API 30
Pixel XL API 30
Resizable (Experimental) API Tiramisu
Select Multiple Devices...
Pair Devices Using Wi-Fi
Wear OS emulator pairing assistant
Device Manager
Troubleshoot Device Connections

**Figure .** The target device menu.

Click **Run**. The emulator might take a minute or so to launch for the first time, but subsequent launches use a snapshot and should launch faster. If you experience issues, see the troubleshooting guide.

Once your app is installed on your AVD, you can run it from the device as you would run any app on a device. Any time you want to deploy new changes, you need to click **Run** or **Apply Changes** again.

## 1.4 Android Apps Development Environment

Android Studio is the official IDE (Integrated Development Environment) for Android application development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- GitHub and Code template integration to assist you to develop common app features and import sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine and many more.

### Create a project

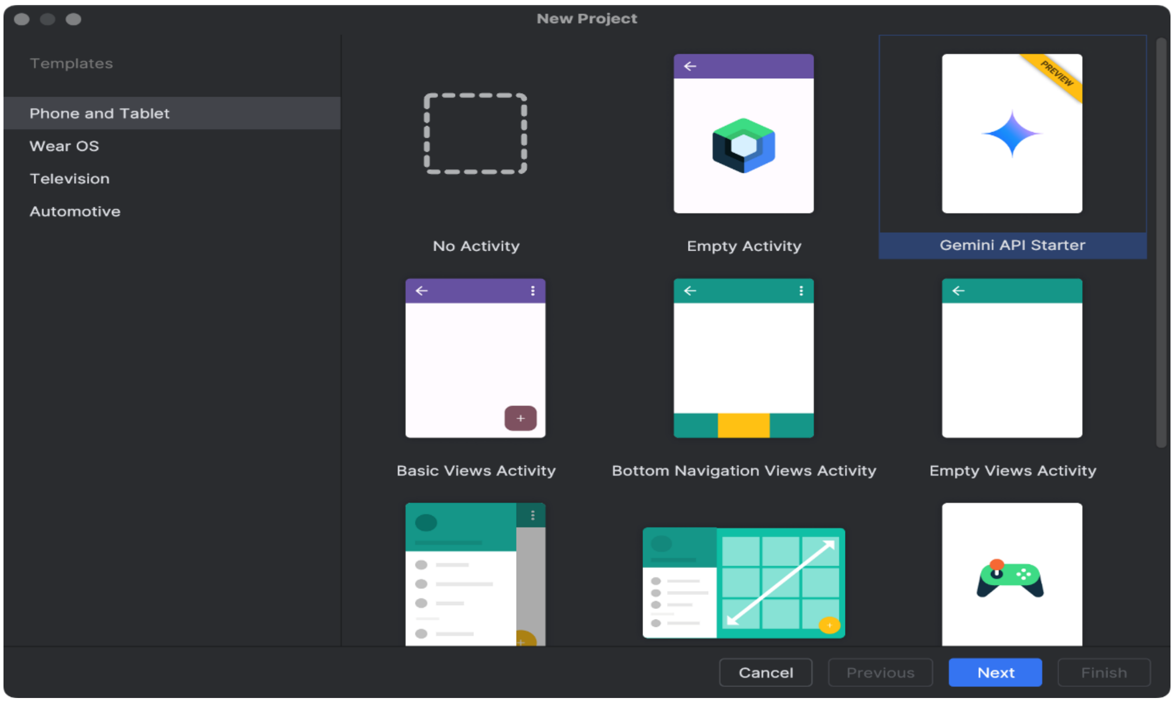
Android Studio makes it easy to create Android apps for various form factors, such as phones, tablets, TVs, and Wear devices. This page explains how to start a new Android app project or import an existing project.

If you do not have a project opened, create a new project by clicking **New Project** on the Android Studio Welcome screen.

If you do have a project opened, create a new project by selecting **File > New > New Project** from the main menu.

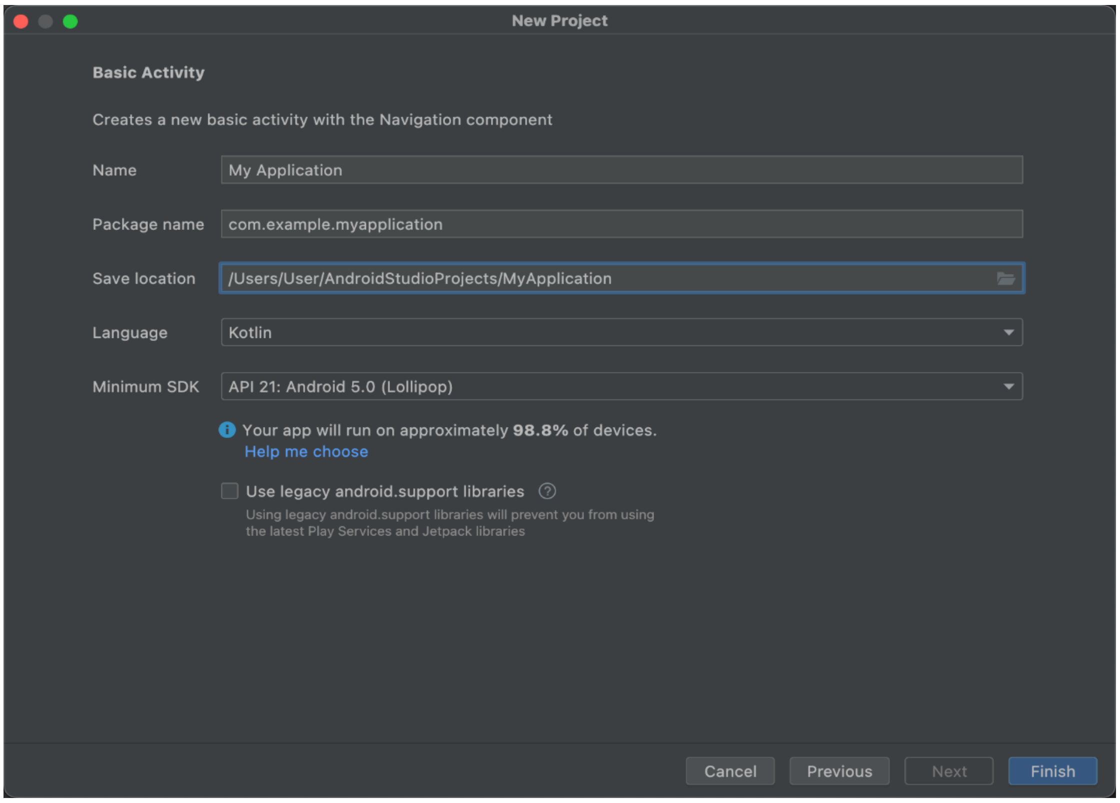
### Choose your project type

In the **New Project** screen that appears, you can select the type of project you want to create from categories of device form factors, shown in the **Templates** pane. For example, figure 1 shows the project templates for phone and tablet.



Selecting the type of project, you want to create lets Android Studio include sample code and resources in your project to help you get started.
Once you select your project type, click **Next**.

**Configure your project**
The next step in creating your project is to configure some settings, as shown in figure 2. If you're creating a **Native C++** project, read Create a new project with C/C++ support to learn more about the options you need to configure.



Basic Activity

Creates a new basic activity with the Navigation component

Name: My Application

Package name: com.example.myapplication

Save location: /Users/User/AndroidStudioProjects/MyApplication

Language: Kotlin

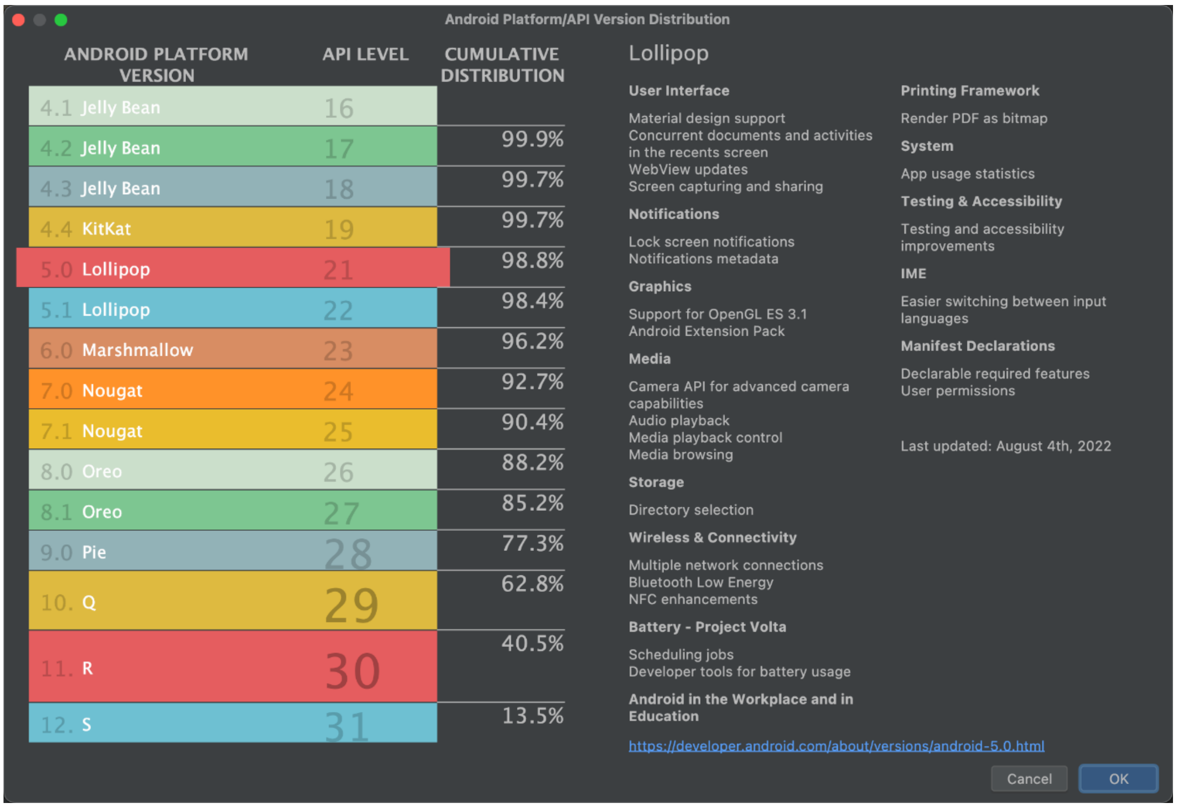
Minimum SDK: API 21: Android 5.0 (Lollipop)

Your app will run on approximately **98.8%** of devices.
Help me choose

☐ Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

- Specify the **Name** of your project.
- Specify the **Package name**. By default, this package name becomes your project's namespace (used to access your project resources) and your project's application ID (used as the ID for publishing). To learn more, see Configure the app module.
- Specify the **Save location** where you want to locally store your project.
- Select the **Language**, Kotlin or Java, you want Android Studio to use when creating sample code for your new project. Keep in mind that you aren't limited to using only that language in the project.
- Select the **Minimum API level** you want your app to support. When you select a lower API level, your app can't use as many modern Android APIs. However, a larger percentage of Android devices can run your app. The opposite is true when selecting a higher API level.

If you want to see more data to help you decide, click **Help me choose**. This displays a dialog showing the cumulative distribution for the API level you have selected and lets you see the impact of using different minimum API levels.



Android Platform/API Version Distribution

| ANDROID PLATFORM VERSION | API LEVEL | CUMULATIVE DISTRIBUTION |
|---|---|---|
| 4.1 Jelly Bean | 16 | |
| 4.2 Jelly Bean | 17 | 99.9% |
| 4.3 Jelly Bean | 18 | 99.7% |
| 4.4 KitKat | 19 | 99.7% |
| 5.0 Lollipop | 21 | 98.8% |
| 5.1 Lollipop | 22 | 98.4% |
| 6.0 Marshmallow | 23 | 96.2% |
| 7.0 Nougat | 24 | 92.7% |
| 7.1 Nougat | 25 | 90.4% |
| 8.0 Oreo | 26 | 88.2% |
| 8.1 Oreo | 27 | 85.2% |
| 9.0 Pie | 28 | 77.3% |
| 10. Q | 29 | 62.8% |
| 11. R | 30 | 40.5% |
| 12. S | 31 | 13.5% |

Lollipop

**User Interface**

Material design support
Concurrent documents and activities in the recents screen
WebView updates
Screen capturing and sharing

**Notifications**

Lock screen notifications
Notifications metadata

**Graphics**

Support for OpenGL ES 3.1
Android Extension Pack

**Media**

Camera API for advanced camera capabilities
Audio playback
Media playback control
Media browsing

**Storage**

Directory selection

**Wireless & Connectivity**

Multiple network connections
Bluetooth Low Energy
NFC enhancements

**Battery - Project Volta**

Scheduling jobs
Developer tools for battery usage

**Android in the Workplace and in Education**

https://developer.android.com/about/versions/android-5.0.html

**Printing Framework**

Render PDF as bitmap

**System**

App usage statistics

**Testing & Accessibility**

Testing and accessibility improvements

**IME**

Easier switching between input languages

**Manifest Declarations**

Declarable required features
User permissions

Last updated: August 4th, 2022

- Your project is configured to use AndroidX libraries by default, which replace the Android Support libraries. To use the legacy support libraries instead, select **Use legacy android.support libraries**. However, this is not recommended, as the legacy support libraries are no longer supported. To learn more, read the AndroidX overview.
- When you're ready to create your project, click **Finish**.

Android Studio creates your new project with some basic code and resources to get you started. If you decide to add support for a different device form factor later, you can add a module to your project. And if you want to share code and resources between modules, you can do so by creating an Android library.

**Gradle-based build system**

Android Studio’s build system is based on Gradle and uses build configuration files written in either Groovy or Kotlin script for ease of extensibility and customization.

Gradle-based projects offer significant features for Android development, including the following:

- Support for binary libraries (AARs). You no longer need to copy library sources into your own projects; you can declare a dependency and the library is automatically downloaded and merged into your project. This includes automatically merging in resources, manifest entries, Proguard exclusion rules, custom lint rules, and so on at build time.
- Support for build variants, which let you build different versions of your app (such as a free version and a pro version) from the same project.
- Easy build configuration and customization. For example, you can pull version names and version codes from Git tags as part of the build.
- Gradle can be used from the IDE, from the command line, and from continuous integration servers like Jenkins, providing the same build everywhere, every time.

For more information about using and configuring Gradle, see Configure your build.

**Dependencies**

Library dependencies in Android Studio use Gradle dependency declarations and Maven dependencies for well-known local source and binary libraries with Maven coordinates. For more information, see Declare dependencies.

**Migrate from IntelliJ**

If your IntelliJ project uses the Gradle build system, you can import your project directly into Android Studio. If your IntelliJ project uses Maven or another build system, you need to set it up to work with Gradle before you can migrate to Android Studio.

**Import a Gradle-based IntelliJ project**

If you are already using Gradle with your IntelliJ project, open it in Android Studio using the following steps:

- Click **File > New > Import Project**.
- Select your IntelliJ project directory and click **OK**. Your project opens in Android Studio.

**Import a non-Gradle IntelliJ project**

If your IntelliJ project doesn't already use the Gradle build system, you have two options for importing your project into Android Studio, which are described in the sections that follow:

- Create a new empty Android Studio project and copy your existing source code into the directories associated with the new project. For more information, see the section about migrating by creating a new empty project.

- Create a new Gradle build file for your project and then import the project and new build file into Android Studio. For more information, see the section about migrating by creating a custom Gradle build file.

**Migrate by creating a new empty project**

To migrate your project into Android Studio by creating a new empty project and copying your source files into the new directories, proceed as follows:

- Open Android Studio and click **File** > **New** > **New Project**.
- Enter a name for your app project and specify the location where it should be created, then click **Next**.
- Select the form factors your app runs on, then click **Next**.
- Click **Add No Activity**, then click **Finish**.
- In the **Project** tool window, click the arrow to open the view menu and select the **Project** view to see and explore the organization of your new Android Studio project. To read more about changing views and how Android Studio structures projects, see Project files.
- Navigate to the location you selected for your new project and move the code, unit tests, instrumentation tests, and resources from your old project directories into the correct locations in your new project structure.
- In Android Studio, click **File** > **Project Structure** to open the Project Structure dialog. Ensure that your app's module is selected in the left pane.
- Make any necessary modifications in the **Properties** tab for your project (for example, modifying the minSdk or targetSdk).
- Click **Dependencies** and add any libraries your project depends on as Gradle dependencies. To add a new dependency, click **Add**, then select the type of dependency you would like to add and follow the prompts.
- Click **OK** to save your modifications.
- Click **Build** > **Make Project** to test building your project, and if necessary resolve any outstanding errors.

**Migrate by creating a custom Gradle build file**

To migrate your project into Android Studio by creating a new Gradle build file to point to your existing source files, proceed as follows:

- Before you begin, back up your project files in a separate location, as the migration process modifies the contents of your project in place.
- Create a file in your project directory called build.gradle, if you're using Groovy, or build.gradle.kts, if you're using Kotlin script. This file contains all the information required for Gradle to run your build.

**Configure Android Studio**

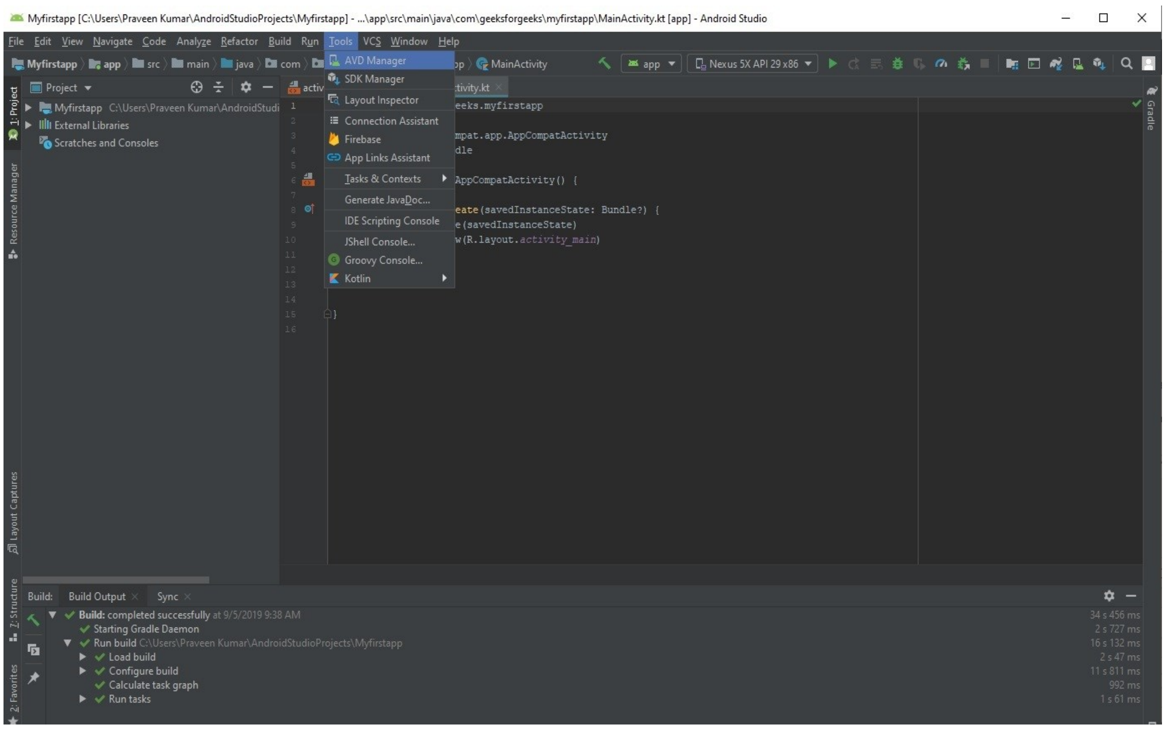
Android Studio provides wizards and templates that verify your system requirements, such as the Java Development Kit (JDK) and available RAM, and configure default settings, such as an optimized default Android Virtual Device (AVD) emulation and updated system images. This document describes additional configuration settings to customize your use of Android Studio.

**Create and manage virtual devices**

In android development, we need an android device to run the application. So, developers of Android Studio provide an option to install android virtual device to run it. In this article, we will learn how to install Android Virtual Device (AVD).

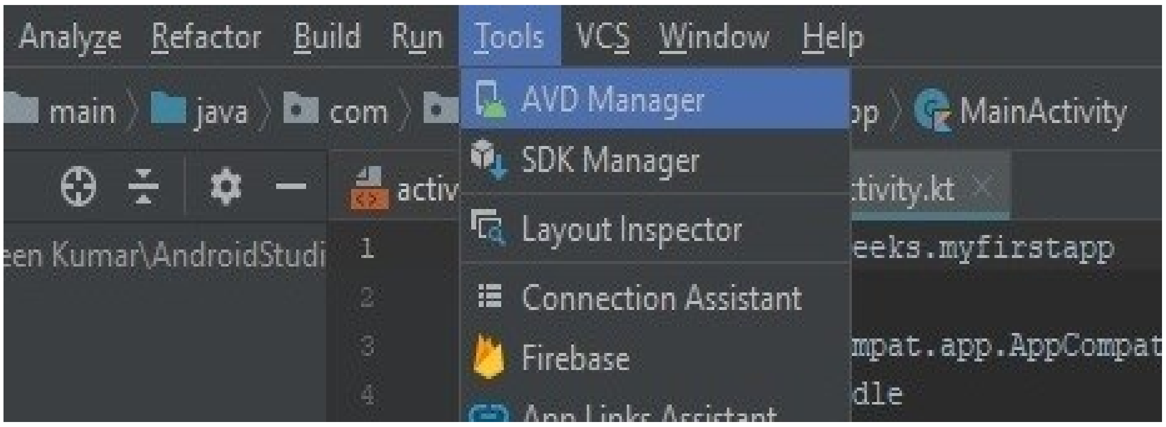
Follow the below steps to install **Android Virtual Device**.

**Step 1:** Go to **Tools** > **AVD Manager**.

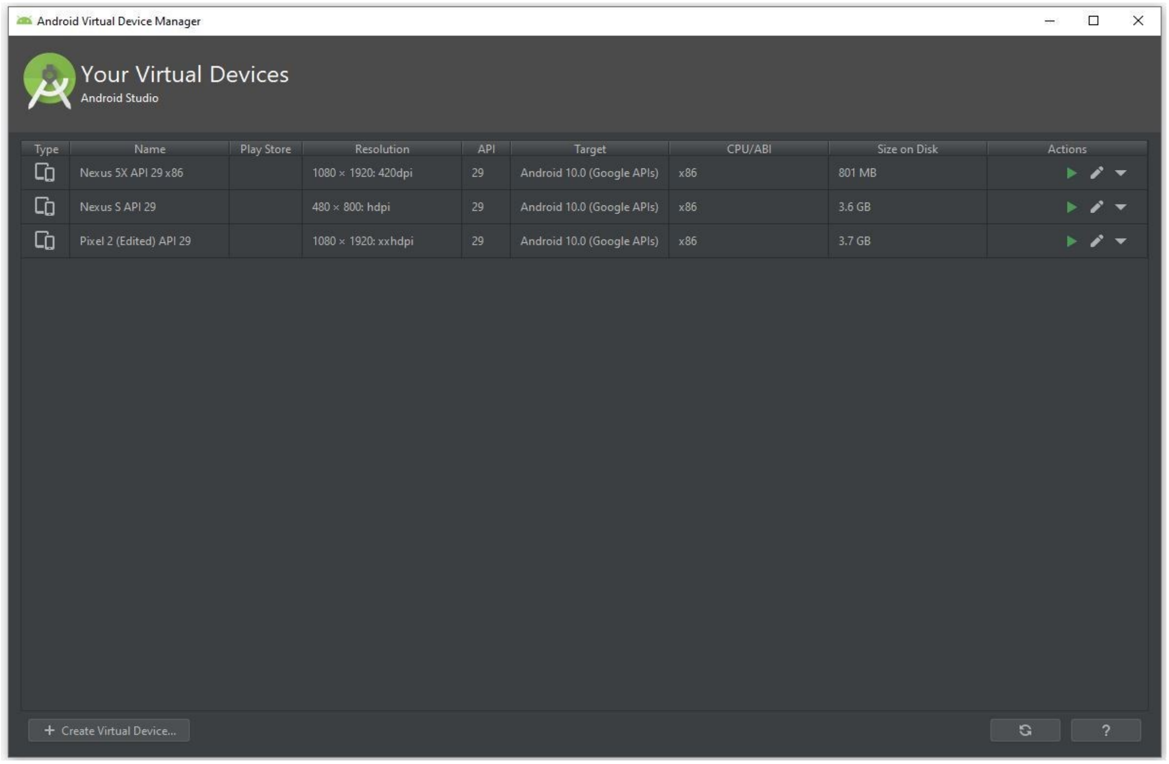

Myfirstapp [C:\Users\Praveen Kumar\AndroidStudioProjects\Myfirstapp] - ...\app\src\main\java\com\geeksforgeeks\myfirstapp\MainActivity.kt [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AVD Manager
SDK Manager
Layout Inspector
Connection Assistant
Firebase
App Links Assistant
Tasks & Contexts
Generate JavaDoc...
IDE Scripting Console
JShell Console...
Groovy Console...
Kotlin
MainActivity
app
Nexus 5X API 29 x86

```
eeks.myfirstapp

mpat.app.AppCompatActivity
dle

AppCompatActivity() {

eate(savedInstanceState: Bundle?) {
e(savedInstanceState)
w(R.layout.activity_main)

}
```

```
Build: completed successfully at 9/5/2019 9:38 AM                              34 s 456 ms
  Starting Gradle Daemon                                                        2 s 727 ms
  Run build C:\Users\Praveen Kumar\AndroidStudioProjects\Myfirstapp             16 s 132 ms
    Load build                                                                  2 s 47 ms
    Configure build                                                             11 s 811 ms
    Calculate task graph                                                        992 ms
    Run tasks                                                                   1 s 61 ms
```



**Step 2:** Now click on **Create Virtual Device**.

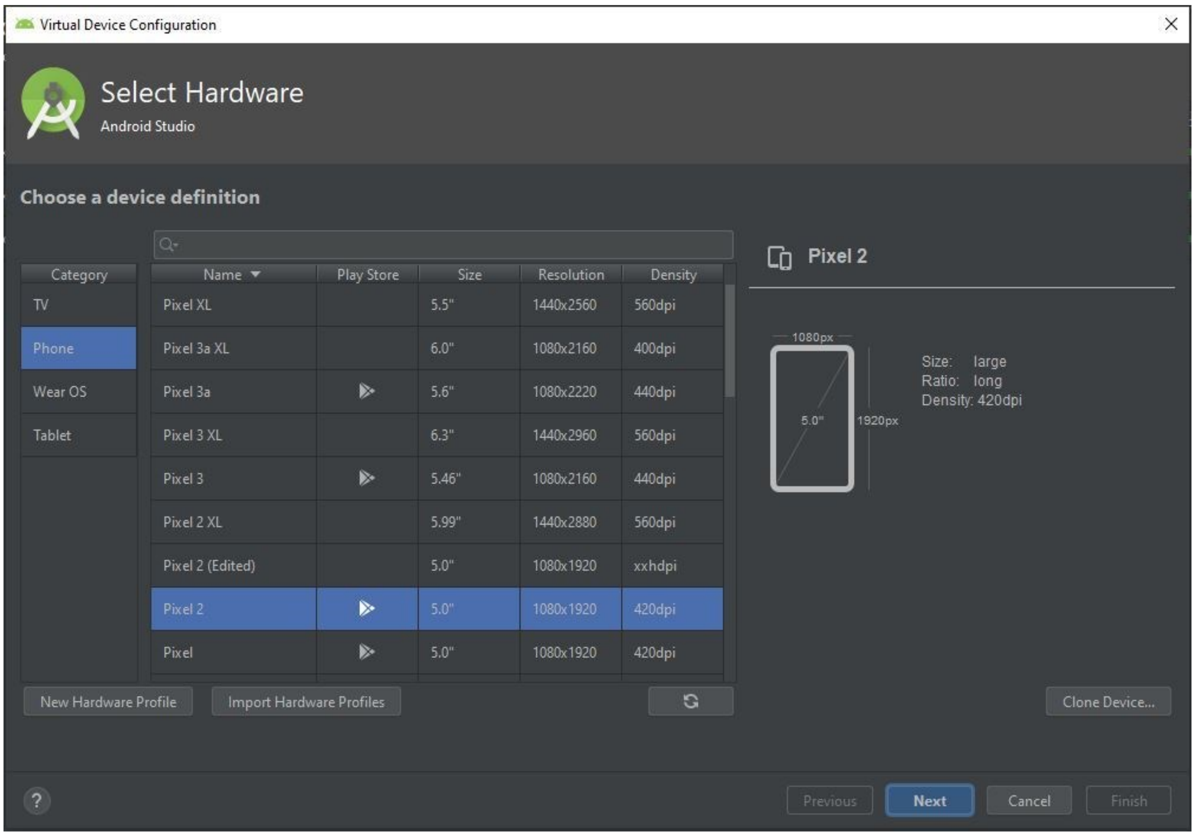


Android Virtual Device Manager

Your Virtual Devices
Android Studio

| Type | Name | Play Store | Resolution | API | Target | CPU/ABI | Size on Disk | Actions |
|---|---|---|---|---|---|---|---|---|
| | Nexus 5X API 29 x86 | | 1080 × 1920: 420dpi | 29 | Android 10.0 (Google APIs) | x86 | 801 MB | |
| | Nexus S API 29 | | 480 × 800: hdpi | 29 | Android 10.0 (Google APIs) | x86 | 3.6 GB | |
| | Pixel 2 (Edited) API 29 | | 1080 × 1920: xxhdpi | 29 | Android 10.0 (Google APIs) | x86 | 3.7 GB | |

**Step 3:** A pop-up window will be there and here we select the category Phone because we are creating android app for mobile and select the model of mobile phone we want to install.



Virtual Device Configuration

Select Hardware
Android Studio

Choose a device definition

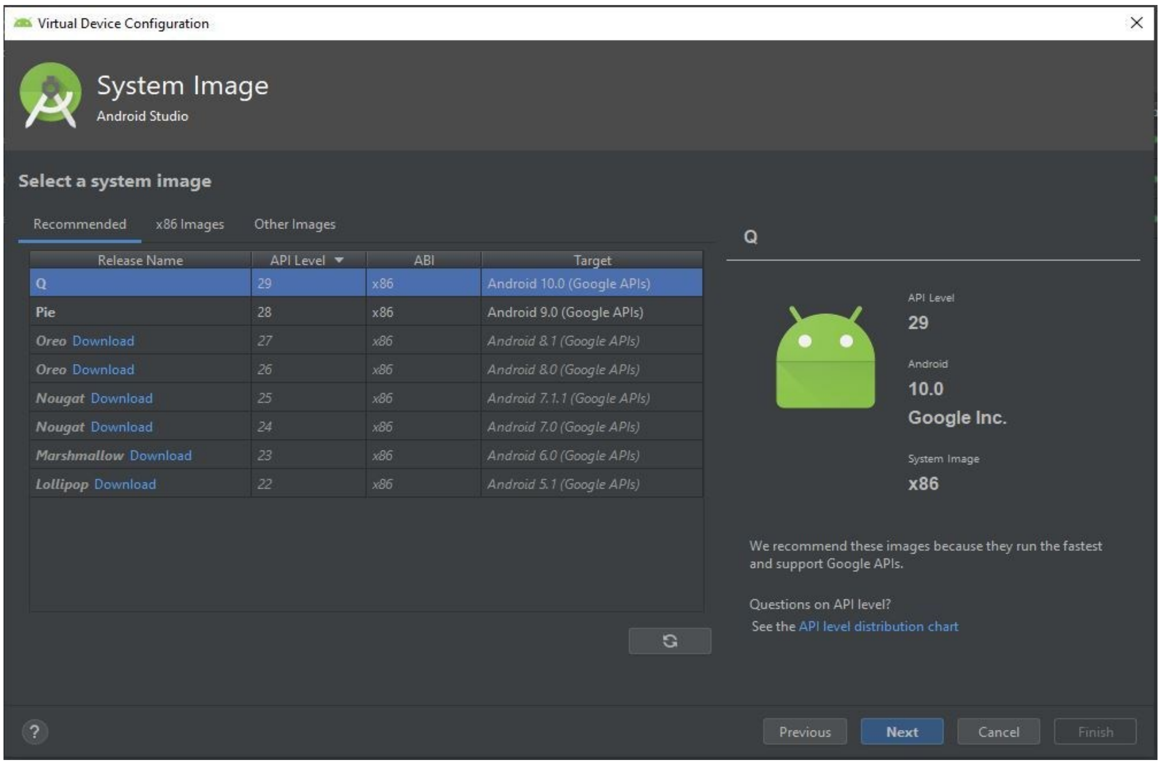
| Category |
|---|
| TV |
| Phone |
| Wear OS |
| Tablet |

| Name | Play Store | Size | Resolution | Density |
|---|---|---|---|---|
| Pixel XL | | 5.5" | 1440x2560 | 560dpi |
| Pixel 3a XL | | 6.0" | 1080x2160 | 400dpi |
| Pixel 3a | ▶ | 5.6" | 1080x2220 | 440dpi |
| Pixel 3 XL | | 6.3" | 1440x2960 | 560dpi |
| Pixel 3 | ▶ | 5.46" | 1080x2160 | 440dpi |
| Pixel 2 XL | | 5.99" | 1440x2880 | 560dpi |
| Pixel 2 (Edited) | | 5.0" | 1080x1920 | xxhdpi |
| Pixel 2 | ▶ | 5.0" | 1080x1920 | 420dpi |
| Pixel | ▶ | 5.0" | 1080x1920 | 420dpi |

Pixel 2

Size: large
Ratio: long
Density: 420dpi

**Step 4: Here we select the android version to download like Q, Pie, Oreo etc and click Next button.**



Virtual Device Configuration

System Image
Android Studio

Select a system image

Recommended | x86 Images | Other Images

| Release Name | API Level | ABI | Target |
|---|---|---|---|
| Q | 29 | x86 | Android 10.0 (Google APIs) |
| Pie | 28 | x86 | Android 9.0 (Google APIs) |
| Oreo Download | 27 | x86 | Android 8.1 (Google APIs) |
| Oreo Download | 26 | x86 | Android 8.0 (Google APIs) |
| Nougat Download | 25 | x86 | Android 7.1.1 (Google APIs) |
| Nougat Download | 24 | x86 | Android 7.0 (Google APIs) |
| Marshmallow Download | 23 | x86 | Android 6.0 (Google APIs) |
| Lollipop Download | 22 | x86 | Android 5.1 (Google APIs) |

Q

API Level
29

Android
10.0
Google Inc.

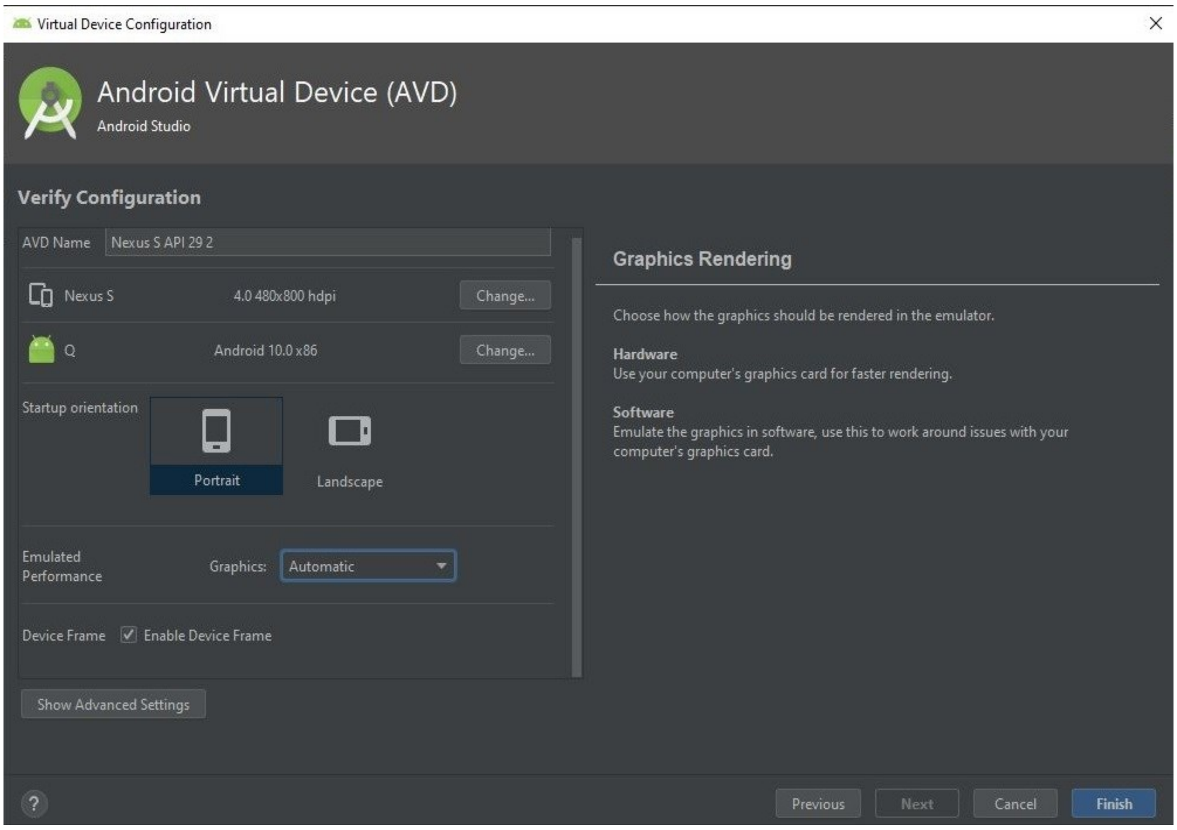
System Image
x86

We recommend these images because they run the fastest and support Google APIs.

Questions on API level?
See the API level distribution chart

Previous | Next | Cancel | Finish

**Step 5: Click the finish button to complete the installation.**



Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Nexus S API 29 2

Nexus S | 4.0 480x800 hdpi | Change...

Q | Android 10.0 x86 | Change...

Startup orientation: Portrait | Landscape

Emulated Performance — Graphics: Automatic

Device Frame — ✓ Enable Device Frame

Show Advanced Settings

Graphics Rendering

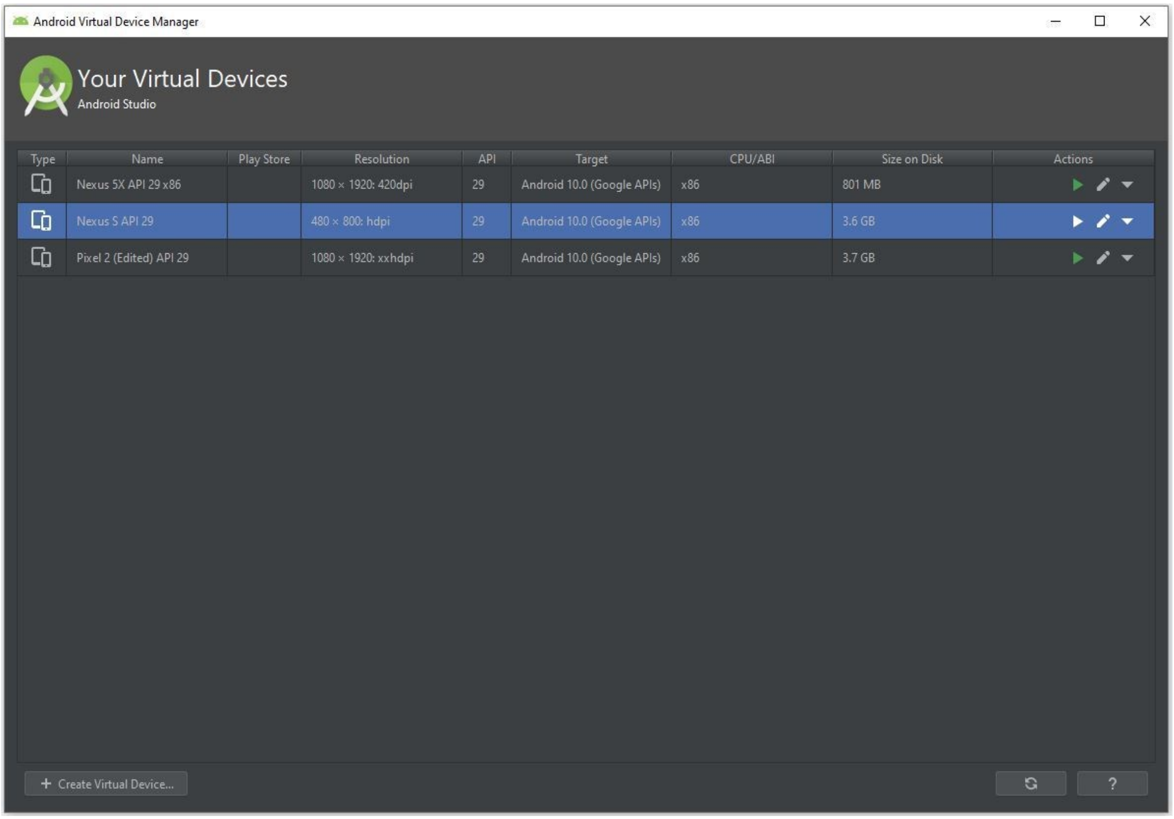
Choose how the graphics should be rendered in the emulator.

Hardware
Use your computer's graphics card for faster rendering.

Software
Emulate the graphics in software, use this to work around issues with your computer's graphics card.

Previous | Next | Cancel | Finish

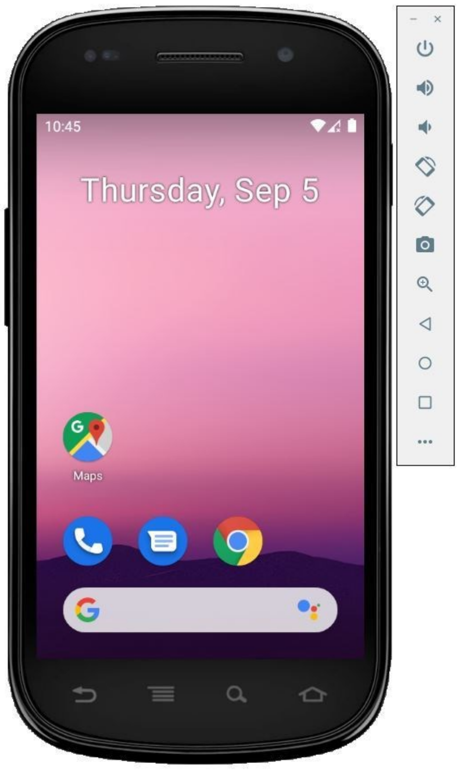
**Step 6: Now we can select the virtual device we want to run as emulator can click on the run icon.**


Android Virtual Device Manager

Your Virtual Devices
Android Studio

| Type | Name | Play Store | Resolution | API | Target | CPU/ABI | Size on Disk | Actions |
|---|---|---|---|---|---|---|---|---|
| | Nexus 5X API 29 x86 | | 1080 × 1920: 420dpi | 29 | Android 10.0 (Google APIs) | x86 | 801 MB | |
| | Nexus S API 29 | | 480 × 800: hdpi | 29 | Android 10.0 (Google APIs) | x86 | 3.6 GB | |
| | Pixel 2 (Edited) API 29 | | 1080 × 1920: xxhdpi | 29 | Android 10.0 (Google APIs) | x86 | 3.7 GB | |

**Step 7: Finally, our virtual device is ready to run our android app**



## Self-Check 1: Install and Setup Android Development Environment

### Questionnaire

1. What is the history of app development?

   **Answer:**

2. What is an overview of Android app development?

   **Answer:**

3. What constitutes the Android application development environment?

   **Answer:**

4. How do you set up the Android development environment

   **Answer:**

5. What is Android Studio?

   **Answer:**

6. Why is JDK necessary for Android development?

   **Answer:**

7. What is the Android SDK?

   **Answer:**

8. How do you create an Android Virtual Device (AVD)?

   **Answer:**

9. Why is emulator acceleration important?

   **Answer:**

10. What is the NDK in Android development?

    **Answer:**

11. How does version control benefit Android development?

    **Answer:**

12. Where can you find Android developer documentation?

    **Answer:**

13. What resources does Google provide for Android developers?

    **Answer:**

14. How do you debug Android applications?

    **Answer:**

15. What are the steps to create and run your first Android app?

    **Answer:**

## Answer Key - 1: Install and Setup Android Development Environment

1. What is the history of app development?

   **Answer:** App development began with the rise of smartphones in the late 2000s, evolving from basic utilities to sophisticated applications across various platforms.

2. What is an overview of Android app development?

   **Answer:** Android app development involves creating software applications that run on Android devices, using the Android SDK and programming languages like Java or Kotlin.

3. What constitutes the Android application development environment?

   **Answer:** The Android development environment includes tools like Android Studio (IDE), the Android SDK, emulators, and physical devices for testing.

4. How do you set up the Android development environment

   **Answer:** Install Java Development Kit (JDK), download and install Android Studio, set up Android SDK components through Android Studio, and configure an Android Virtual Device (AVD).

5. What is Android Studio?

   **Answer:** Android Studio is the official IDE for Android development, based on IntelliJ IDEA, providing tools for coding, testing, and debugging Android apps.

6. Why is JDK necessary for Android development?

   **Answer:** Android apps are primarily developed using Java (or Kotlin), hence JDK is required for compiling and running Java code.

7. What is the Android SDK?

   **Answer:** The Android SDK (Software Development Kit) is a set of tools, libraries, and documentation necessary for developing Android applications.

8. How do you create an Android Virtual Device (AVD)?

   **Answer:** Use the AVD Manager in Android Studio to create virtual devices with specific configurations (e.g., screen size, Android version) for testing apps.

9. Why is emulator acceleration important?

**Answer:** Emulator acceleration (using technologies like HAXM or Intel VT-x) speeds up the Android Emulator, making app testing more efficient.

10. What is the NDK in Android development?

    **Answer:** The NDK (Native Development Kit) allows developers to build performance-critical portions of their apps in native code (C or C++).

11. How does version control benefit Android development?

    **Answer:** Version control (e.g., Git) helps manage changes to source code, facilitating collaboration, and enabling rollback to previous versions if needed.

12. Where can you find Android developer documentation?

    **Answer:** Android developer documentation is available at [developer.android.com](https://developer.android.com).

13. What resources does Google provide for Android developers?

    **Answer:** Google offers tutorials, guides, sample code, and the Android Developer YouTube channel to support developers.

14. How do you debug Android applications?

    **Answer:** Android Studio provides tools like logcat, breakpoints, and a debugger interface to identify and fix issues in Android apps.

15. What are the steps to create and run your first Android app?

    **Answer:** Create a new project in Android Studio, design the user interface, write code to implement functionality, debug as needed, and run the app on a virtual device or physical device.

## Job Sheet-1.1: Install JDK and Set  Environment Variables

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Install the latest version of the Java Development Kit (JDK) on your computer.
6. Set the JAVA_HOME environment variable to the installation directory of the JDK.
7. Update the PATH environment variable to include the bin directory of the JDK.
8. Show your work to the trainer.
9. Clean your Work Place as per standard procedure.
10. Turn off the computer and clean your workplace.

## Specification Sheet-1.1: Install JDK and Set Environment Variables

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet-1.2: Download and Install Android Studio

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Download the latest version of Android Studio from the official Android Studio Download page.
6. Follow the installation instructions for your operating system (Windows, macOS, or Linux).
7. Launch Android Studio after installation and complete the initial setup.
8. Show your work to the trainer.
9. Clean your work place as per standard procedure.
10. Turn off the computer and clean your workplace.

## Specification Sheet-1.2: Download and Install Android Studio

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection | | … | 1 |

## Job Sheet-1.3: Configure Android SDK

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and access the SDK Manager.
6. Download the necessary Android SDK components, including the Android SDK Platform and Android Virtual Device (AVD) components.
7. Configure the Android SDK location in Android Studio.
8. Show your work to the trainer.
9. Clean your work place as per standard procedure.
10. Turn off the computer and clean your workplace.

## Specification Sheet-1.3: Configure Android SDK

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet-1.4: Set Up an Android Virtual Device (AVD)

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Use the AVD Manager in Android Studio to create a virtual device for testing.
6. Choose a target Android version and device configuration.
7. Launch the virtual device and test its functionality. Show your work to the trainer.
8. Clean your work place as per standard procedure.
9. Turn off the computer and clean your workplace.

## Specification Sheet-1.4: Set Up an Android Virtual Device (AVD)

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet-1.5: Connect a Physical Android Device

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. If you have a physical Android device, enable Developer Options and USB Debugging.
6. Connect your device to the computer via USB and ensure that it is recognized by Android Studio.
7. Verify that you can deploy and run an app on the physical device.
8. Show your work to the trainer.
9. Clean your work place as per standard procedure.
10. Turn off the computer and clean your workplace.

## Specification Sheet-1.5: Connect a Physical Android Device

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection | | … | 1 |

## Job Sheet-1.6: Create a New Android Project

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and create a new Android project.
6. Choose a project template and configure project settings.
7. Write a simple "Hello World" application and run it on the emulator or a physical device.
8. Run the app and Show your work to the trainer.
9. Clean your work place as per standard procedure.
10. Turn off the computer and clean your workplace.

## Specification Sheet-1.6: Create a New Android Project

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet-1.7: Perform Version Control Integration

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and click on "Start a new Android Studio project".
6. Choose "Empty Activity" and name the project "MyUIApp".
7. Choose the language as Kotlin and click "Next".
8. Choose the minimum SDK version as API 21 and click "Finish"
9. Set up version control for your Android project using Git.
10. Initialize a Git repository for your project and make an initial commit.
11. Use Android Studio's version control features to track changes.
12. Show your work to the trainer.
13. Clean your work place as per standard procedure.
14. Turn off the computer and clean your workplace.

## Specification Sheet-1.7: Perform Version Control Integration

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection | | … | 1 |

## Job Sheet-1.8: Explore Android Studio Features

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and click on "Start a new Android Studio project".
6. Choose "Empty Activity" and name the project "MyUIApp".
7. Choose the language as Kotlin and click "Next".
8. Choose the minimum SDK version as API 21 and click "Finish"
9. Explore and experiment with various features of Android Studio, such as code editor, layout editor, debugging tools, and performance analysis tools.
10. Create a simple UI using the layout editor and associate it with the application logic.
11. Show your work to the trainer.
12. Clean your work place as per standard procedure.
13. Turn off the computer and clean your workplace.

## Specification Sheet-1.8: Explore Android Studio Features

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet-1.9: Deploy and Test on Multiple Devices

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and click on "Start a new Android Studio project".
6. Choose "Empty Activity" and name the project "MyApp".
7. Choose the language as Kotlin and click "Next".
8. Choose the minimum SDK version as API 21 and click "Finish"
9. Test your Android app on multiple virtual devices with different screen sizes and resolutions.
10. If possible, test the app on a physical Android device with different specifications.
11. Show your work to the trainer.
12. Clean your work place as per standard procedure.
13. Turn off the computer and clean your workplace.

## Specification Sheet-1.9: Deploy and Test on Multiple Devices

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection | | … | 1 |

## Job Sheet-1.10: Perform Gradle Build Configuration

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and click on "Start a new Android Studio project".
6. Choose "Empty Activity" and name the project "MyUIApp".
7. Choose the language as Kotlin and click "Next".
8. Choose the minimum SDK version as API 21 and click "Finish"
9. Understand the Gradle build system used by Android Studio.
10. Modify the build gradle file of your project to include additional dependencies or configurations.
11. Sync the project with Gradle and observe the changes.
12. Show your work to the trainer.
13. Clean your work place as per standard procedure.
14. Turn off the computer and clean your workplace.

## Specification Sheet-1.10: Perform Gradle Build Configuration

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Learning Outcome 2: Use main building blocks

| | |
|---|---|
| Assessment Criteria | 1. Main building block of Android apps is used.<br>2. Activity life cycle is used.<br>3. UI Widgets are used.<br>4. View and Layout are created |
| Conditions and Resources | • Actual workplace or training environment<br>• CBLM<br>• Handouts<br>• Job related tools, equipment, and materials<br>• Multimedia Projector<br>• Paper, Pen, Pencil, and Eraser<br>• Internet Facilities<br>• Whiteboard and Marker |
| Contents | 1. Main building block of Android apps<br>2. Use activity life cycle<br>3. UI Widgets<br>4. View and Layout<br>▪ Linear o Relative<br>▪ Table<br>▪ Frame<br>▪ Constraint layout |
| Activities/job/Task | 1. Use Activity Lifecycle with following activity<br>a. Create a new Android project with a single MainActivity.<br>b. Implement the key methods of the Activity lifecycle (onCreate, onStart, onResume, onPause, onStop, onDestroy).<br>c. Print log messages in each method to observe the sequence of lifecycle events.<br>d. Add a button to the layout and handle its click event in the MainActivity<br>2. UI Widgets and Layout with following activity<br>a. Design a simple UI using various UI widgets like TextView, Button, EditText, and ImageView.<br>b. Create a layout (e.g., LinearLayout or RelativeLayout) to arrange these widgets.<br>c. Customize the appearance of widgets using attributes like text color, background color, etc.<br>d. Implement functionality to update the TextView content when a button is clicked.<br>3. Create view and Layout with following activity: |

| | |
|---|---|
| | a. Create a new Android project with a blank MainActivity.<br>b. Define a custom view class (e.g., Custom View) that extends View.<br>c. Override the onDraw method in the custom view class to draw a simple shape (e.g., a rectangle or circle).<br>d. In the Main Activity, add an instance of the custom view to the layout and observe the drawn shape. |
| Training Methods | • Blended<br>• Discussion<br>• Presentation<br>• Demonstration<br>• Guided Practice<br>• Individual Practice<br>• Project Work<br>• Problem Solving<br>• Brainstorming |
| Assessment Methods | Assessment methods may include but not limited to<br>• Written Test<br>• Demonstration<br>• Oral Questioning |

## Learning Experience 2: Use main building blocks

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials Use Kotlin |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Use main building blocks" | 2. Read Information sheet 2: Use main building blocks<br>3. Answer Self-check 2: Use main building blocks<br>4. Check your answer with Answer key 2: Use main building blocks |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job/Task Sheet and Specification Sheet<br><br>**Job Sheet 2.1:** Use Activity Lifecycle with following activity<br>a. Create a new Android project with a single MainActivity.<br>b. Implement the key methods of the Activity lifecycle (onCreate, onStart, onResume, onPause, onStop, onDestroy).<br>c. Print log messages in each method to observe the sequence of lifecycle events.<br>d. Add a button to the layout and handle its click event in the MainActivity<br>**Specification Sheet-2.1:** Use Activity Lifecycle with following activity<br><br>**Job Sheet 2.2:** UI Widgets and Layout with following activity<br>a. Design a simple UI using various UI widgets like TextView, Button, EditText, and ImageView.<br>b. Create a layout (e.g., LinearLayout or RelativeLayout) to arrange these widgets.<br>c. Customize the appearance of widgets using attributes like text color, background color, etc.<br>d. Implement functionality to update the TextView content when a button is clicked. |

| | |
|---|---|
| | **Specification Sheet-2.2:** UI Widgets and Layout with following activity<br><br>**Job Sheet 2.3:** Create view and Layout with following activity:<br>a. Create a new Android project with a blank MainActivity.<br>b. Define a custom view class (e.g., Custom View) that extends View.<br>c. Override the onDraw method in the custom view class to draw a simple shape (e.g., a rectangle or circle).<br>d. In the Main Activity, add an instance of the custom view to the layout and observe the drawn shape.<br><br>**Specification Sheet-2.3:** Create view and Layout with following activity |

## Information Sheet 2: Use main building blocks

**Learning Objective:** After completion of this information sheet, the learners will be able to explain, define and interpret the following contents

2.1 Main building block of Android apps
2.2 Use activity life cycle
2.3 UI Widgets
2.4 View and Layout

### 2.1 Main building block of Android apps

The main building blocks are components that you use as an application developer to build Android apps. They are the conceptual items that you put together to create a bigger whole. When you start thinking about your application, it is good to take a top-down approach. You design your application in terms of screens, features, and the interactions between them. You start with conceptual drawing, something that you can represent in terms of "lines and circles." This approach to application development helps you see the big picture—how the components fit together and how it all makes sense.

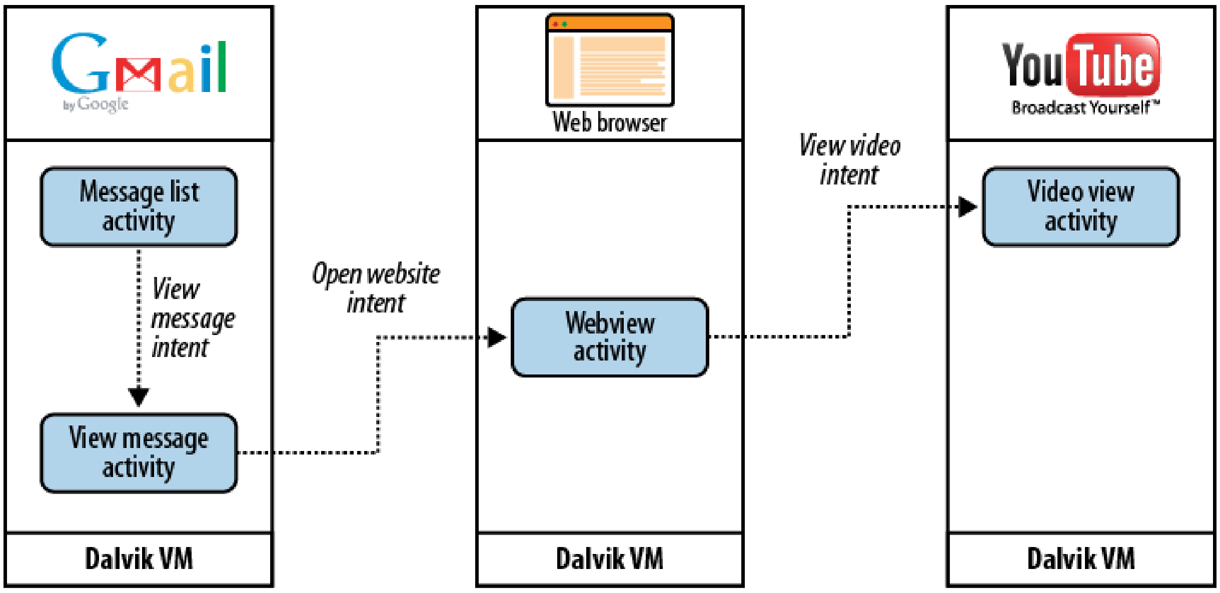
**a. Activities**

An activity is usually a single screen that the user sees on the device at one time. An application typically has multiple activities, and the user flips back and forth among them. As such, activities are the most visible part of your application.
I usually use a website as an analogy for activities. Just like a website consists of multiple pages, so does an Android application consist of multiple activities. Just like a website has a "home page," an Android app has a "main" activity, usually the one that is shown first when you launch the application. And just like a website must provide some sort of navigation among various pages, an Android app should do the same.
On the Web, you can jump from a page on one website to a page on another. Similarly, in Android, you could be looking at an activity of one application, but shortly after you could start another activity in a separate application. For example, if you are in your Contacts app and you choose to text a friend, you would be launching the activity to compose a text message in the Messaging application.

**b. Intents**

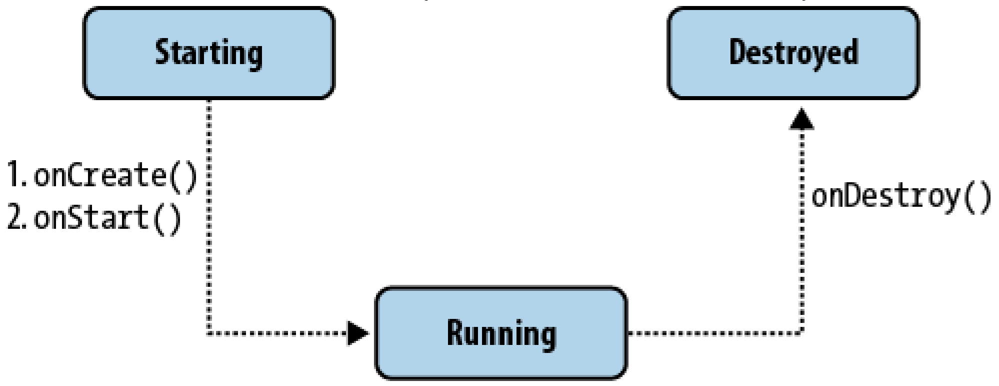
Intents are messages that are sent among the major building blocks. They trigger an activity to start up, tell a service to start or stop, or are simply broadcasts. Intents are asynchronous, meaning the code that sends them doesn't have to wait for them to be completed.
An intent could be explicit or implicit. In an explicit intent, the sender clearly spells out which specific component should be on the receiving end. In an implicit intent, the sender specifies the type of receiver. For example, your activity could send an intent saying it

simply wants someone to open up a web page. In that case, any application that is capable of opening a web page could "compete" to complete this action.
When you have competing applications, the system will ask you which one you'd like to use to complete a given action. You can also set an app as the default. This mechanism works very similarly to your desktop environment, for example, when you downloaded Firefox or Chrome to replace your default Internet Explorer or Safari web browsers.
This type of messaging allows the user to replace any app on the system with a custom one. For example, you might want to download a different SMS application or another browser to replace your existing ones.



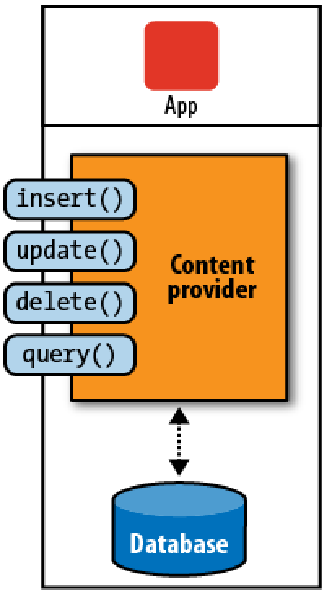
**c. Services**

Services run in the background and don't have any user interface components. They can perform the same actions as activities, but without any user interface. Services are useful for actions that we want to perform for a while, regardless of what is on the screen. For example, you might want your music player to play music even as you are flipping between other applications.
Services have a much simpler life cycle than activities. You either start a service or stop it. Also, the service life cycle is controlled by the developer, and not so much by the system. Consequently, we as developers must be mindful to run our services so that they do not consume shared resources unnecessarily, such as the CPU and battery.



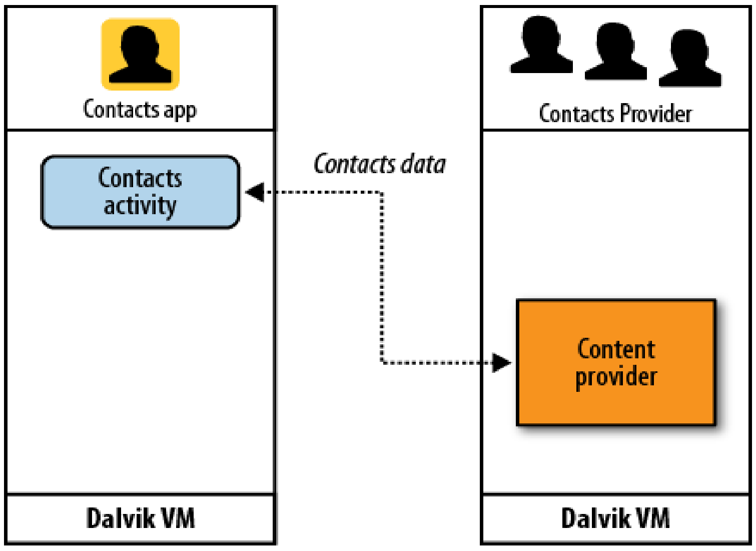
### d. Content Providers

Content providers are interfaces for sharing data between applications. By default, Android runs each application in its own sandbox so that all data that belongs to an application is totally isolated from other applications on the system. Although small amounts of data can be passed between applications via intents, content providers are much better suited for sharing persistent data between possibly large datasets. As such, the content provider API icily adheres to the CRUD principle. illustrates how the content provider's CRUD interface pierces the application boundaries and allows other apps to connect to it to share data.



The Android system uses this mechanism all the time. For example, Contacts Provider is a content provider that exposes all user contact data to various applications. Settings Provider exposes system settings to various applications, including the built-in Settings application. Media Store is responsible for storing and sharing various media, such as photos and music, across various applications. illustrates how the Contacts app uses Contacts Provider, a totally separate application, to retrieve data about users' contacts. The Contacts app itself does not have any contacts data, and Contacts Provider doesn't have any user interface.

This separation of data storage and the actual user interface application offers greater



flexibility to mash up various parts of the system. For example, a user could install an alternative address book application that uses the same data as the default Contacts app. Or, he could install widgets on the home screen that allow for easy changes in the System Settings, such as turning on or off the WiFi, Bluetooth, or GPS features. Many phone manufactures take advantage of content providers to add their own applications on top of standard Android to improve overall user experience, such as HTC Sense.

Content providers are relatively simple interfaces, with the standard insert(), update(), delete(), and query() methods. These methods look a lot like standard database methods, so it is relatively easy to implement a content provider as a proxy to the database. Having said that, you are much more likely to use content providers than write your own.

**e. Broadcast Receivers**

Broadcast receivers are Android's implementation of a system-wide publish/subscribe mechanism, or more precisely, an Observer pattern. The receiver is simply dormant code that gets activated once an event to which it is subscribed happens.

The system itself broadcasts events all the time. For example, when an SMS arrives, a call comes in, the battery runs low, or the system gets booted, all those events are broadcasted, and any number of receivers could be triggered by them.
In our Twitter app example, we want to start the update service once the system starts up. To do that, we can subscribe to the broadcast that tells us the system has completed booting up. You can also send your own broadcasts from one part of your application to another, or to a totally different application.

Broadcast receivers themselves do not have any visual representation, nor are they actively running in memory. But when triggered, they get to execute some code, such as starting an activity, a service, or something else.

**f. Application Context**

So far you have seen activities, services, content providers, and broadcast receivers. Together, they make up an application. Another way of saying this is that they live inside the same application context. Application context refers to the application environment and the process within which all its components are running. It allows applications to share the data and resources between various building blocks. An application context gets created whenever the first component of this application is started up, regardless of whether that component is an activity, service, or something else. Application context lives if your application is alive. As such, it is independent of the activity's life cycle.

**Additional building block of Android apps**

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

| SL .No | Components & Description |
|---|---|
| 1 | **Fragments:** Represents a portion of user interface in an Activity. |
| 2 | **Views:** UI elements that are drawn on-screen including buttons, lists forms etc. |
| 3 | **Layouts:** View hierarchies that control screen format and appearance of the views. |
| 4 | **Resources:** External elements, such as strings, constants and drawable pictures. |
| 5 | **Manifest:** Configuration file for the application. |

## 2.2 Use activity life cycle

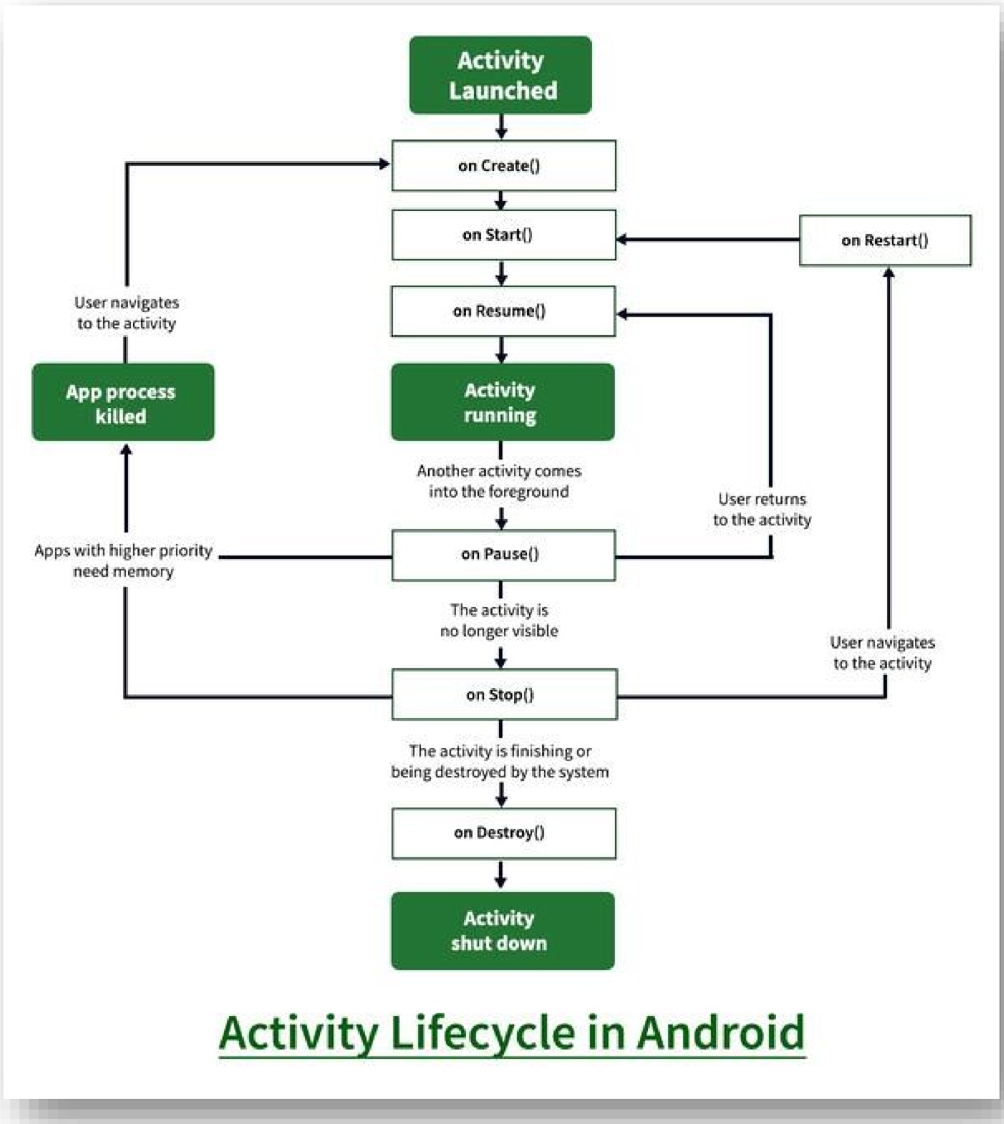
### Activity Life Cycle

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.

Each activity goes through various stages or a lifecycle and is managed by activity stacks. So, when a new activity starts, the previous one always remains below it. There are four stages of an activity.

- If an activity is in the foreground of the screen i.e at the top of the stack, then it is said to be active or running. This is usually the activity that the user is currently interacting with.
- If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity. In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.
- If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.
- The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

For each stage, android provides us with a set of 7 methods that have their own significance for each stage in the life cycle. The image shows a path of migration whenever an app switches from one state to another.


Activity Launched
on Create()
on Start()
on Restart()
on Resume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down
Activity Lifecycle in Android

**a. onCreate()**

It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.

**Example:**

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
```

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)

// Bundle containing previous frozen state
setContentView(R.layout.activity_main)

// The content view pointing to the id of layout
// in the file activity_main.xml
                val toast = Toast.makeText(applicationContext, "onCreate
Called", Toast.LENGTH_LONG).show()
}
}
```

**b. onStart()**

It is invoked when the activity is visible to the user. It is followed by onResume() if the activity is invoked from the background. It is also invoked after onCreate() when the activity is first started.

**Example**:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
                val toast = Toast.makeText(applicationContext, "onCreate
Called", Toast.LENGTH_LONG).show()
        }
override fun onStart() {
        super.onStart()
        // It will show a message on the screen
        // then onStart is invoked
                val toast = Toast.makeText(applicationContext, "onStart
Called", Toast.LENGTH_LONG).show()
        }
}
```

**c. onRestart()**

It is invoked after the activity has been stopped and prior to its starting stage and thus is always followed by onStart() when any activity is revived from background to on-screen.

**Example**:

```
        import android.os.Bundle
        import android.widget.Toast
        import androidx.appcompat.app.AppCompatActivity
        class MainActivity : AppCompatActivity() {
                override fun onCreate(savedInstanceState: Bundle?) {
                        super.onCreate(savedInstanceState)
                        setContentView(R.layout.activity_main)
                        val toast = Toast.makeText(applicationContext, "onCreate
        Called", Toast.LENGTH_LONG).show()
                        }
                override fun onRestart() {
                        super.onRestart()
                        // It will show a message on the screen
                        // then onRestart is invoked
                        val toast = Toast.makeText(applicationContext, "onRestart
        Called", Toast.LENGTH_LONG).show()
                }
                }
```

**d. onResume()**

It is invoked when the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, with a user interacting with it. Always followed by onPause() when the activity goes into the background or is closed by the user.

**Example**:

```
        import android.os.Bundle
        import android.widget.Toast
        import androidx.appcompat.app.AppCompatActivity
        class MainActivity : AppCompatActivity() {
                override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val toast = Toast.makeText(applicationContext, "onCreate Called", To
        ast.LENGTH_LONG).show()
        }
        override fun onResume() {
        super.onResume()
        // It will show a message on the screen
        // then onResume is invoked
        val toast = Toast.makeText(applicationContext, "onResume Called",
       Toast.LENGTH_LONG).show()
                }
        }
```

**e. onPause()**

It is invoked when an activity is going into the background but has not yet been killed. It is a counterpart to onResume(). When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen). The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.

**Example**:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)
                val toast = Toast.makeText(applicationContext, "onCreate
Called", Toast.LENGTH_LONG).show()
                }
        override fun onPause() {
                super.onPause()
                // It will show a message on the screen
                // then onPause is invoked
                val toast = Toast.makeText(applicationContext, "onPause
Called", Toast.LENGTH_LONG).show()
        }
}
```

**f. onStop()**

It is invoked when the activity is not visible to the user. It is followed by **onRestart()** when the activity is revoked from the background, followed by onDestroy() when the activity is closed or finished, and nothing when the activity remains on the background only. Note that this method may never be called, in low memory situations where the system does not have enough memory to keep the activity's process running after its onPause() method is called.

**Example**:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)
```

```
                    val toast = Toast.makeText(applicationContext, "onCreate
        Called", Toast.LENGTH_LONG).show()
                    }
            override fun onStop() {
                    super.onStop()
                    // It will show a message on the screen
                    // then onStop is invoked
                    val toast = Toast.makeText(applicationContext, "onStop
        Called", Toast.LENGTH_LONG).show()
            }
        }
```

**g. onDestroy()**

The final call received before the activity is destroyed. This can happen either because the activity is finishing (when finish() is invoked) or because the system is temporarily destroying this instance of the activity to save space. To distinguish between these scenarios, check it with **isFinishing()** method.

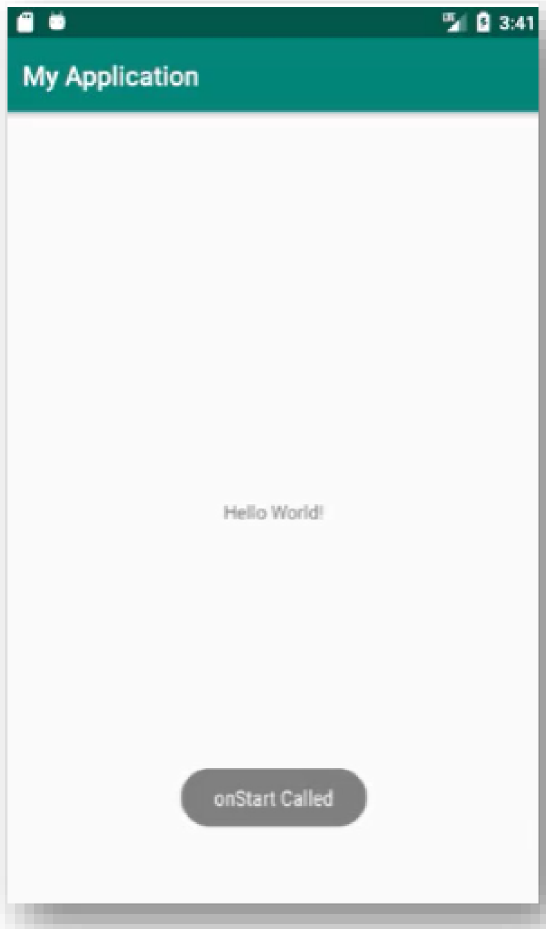
**Example**:

```
        import android.os.Bundle
        import android.widget.Toast
        import androidx.appcompat.app.AppCompatActivity

        class MainActivity : AppCompatActivity() {

            override fun onCreate(savedInstanceState: Bundle?) {
                    super.onCreate(savedInstanceState)
                    setContentView(R.layout.activity_main)

                    val toast = Toast.makeText(applicationContext, "onCreate
        Called", Toast.LENGTH_LONG).show()
                    }

            override fun onDestroy() {
                    super.onDestroy()
                    // It will show a message on the screen
                    // then onDestroy is invoked
                    val toast = Toast.makeText(applicationContext, "onDestroy
        Called", Toast.LENGTH_LONG).show()
            }
        }
```

**Android App to Demonstrate Activity Lifecycle in Android**

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)
                val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
                }
        override fun onStart() {
                super.onStart()
                val toast = Toast.makeText(applicationContext, "onStart Called",
Toast.LENGTH_LONG).show()
        }
        override fun onRestart() {
                super.onRestart()
                val toast = Toast.makeText(applicationContext, "onRestart Called",
Toast.LENGTH_LONG).show()
        }
        override fun onPause() {
                super.onPause()
                val toast = Toast.makeText(applicationContext, "onPause Called",
Toast.LENGTH_LONG).show()
        }
        override fun onResume() {
                super.onResume()
                val toast = Toast.makeText(applicationContext, "onResume Called",
Toast.LENGTH_LONG).show()
        }
        override fun onStop() {
                super.onStop()
                val toast = Toast.makeText(applicationContext, "onStop Called",
Toast.LENGTH_LONG).show()
        }
        override fun onDestroy() {
                super.onDestroy()
                val toast = Toast.makeText(applicationContext, "onDestroy Called",
Toast.LENGTH_LONG).show()
        }
}
```

**Output:**



## 2.3 UI Widgets

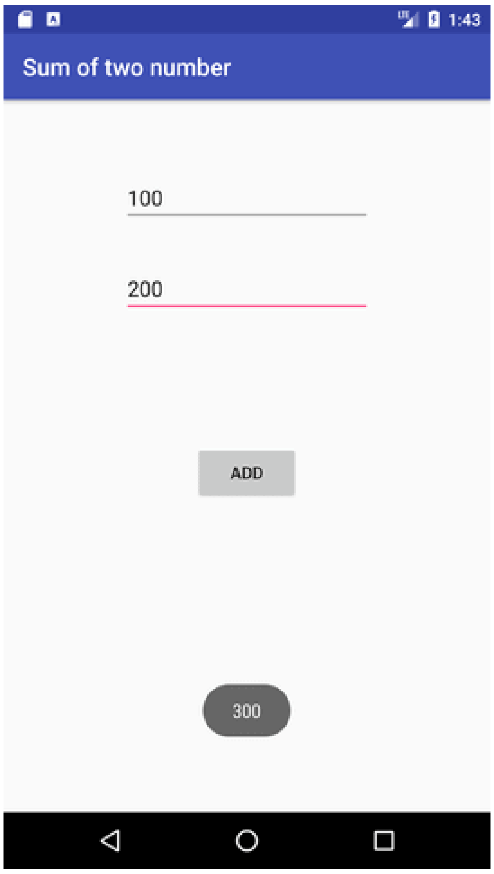
A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

Widgets could be of many types such as information widgets, collection widgets, control widgets and hybrid widgets. Android provides us a complete framework to develop our own widgets.

**Android Button Example**

- Triggers an action when clicked.
- Used for actions like submitting a form, navigating to another screen, etc.

```
package example.javatpoint.com.sumoftwonumber;
 import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
 public class MainActivity extends AppCompatActivity {
   private EditText edittext1, edittext2;
   private Button buttonSum;
     @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
         addListenerOnButton();
   }
     public void addListenerOnButton() {
       edittext1 = (EditText) findViewById(R.id.editText1);
       edittext2 = (EditText) findViewById(R.id.editText2);
       buttonSum = (Button) findViewById(R.id.button);
         buttonSum.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
               String value1=edittext1.getText().toString();
               String value2=edittext2.getText().toString();
               int a=Integer.parseInt(value1);
               int b=Integer.parseInt(value2);
               int sum=a+b;
               Toast.makeText(getApplicationContext(),String.valueOf(sum), Toast.LENGT
H_LONG).show();
           }
       });
   }
}
```



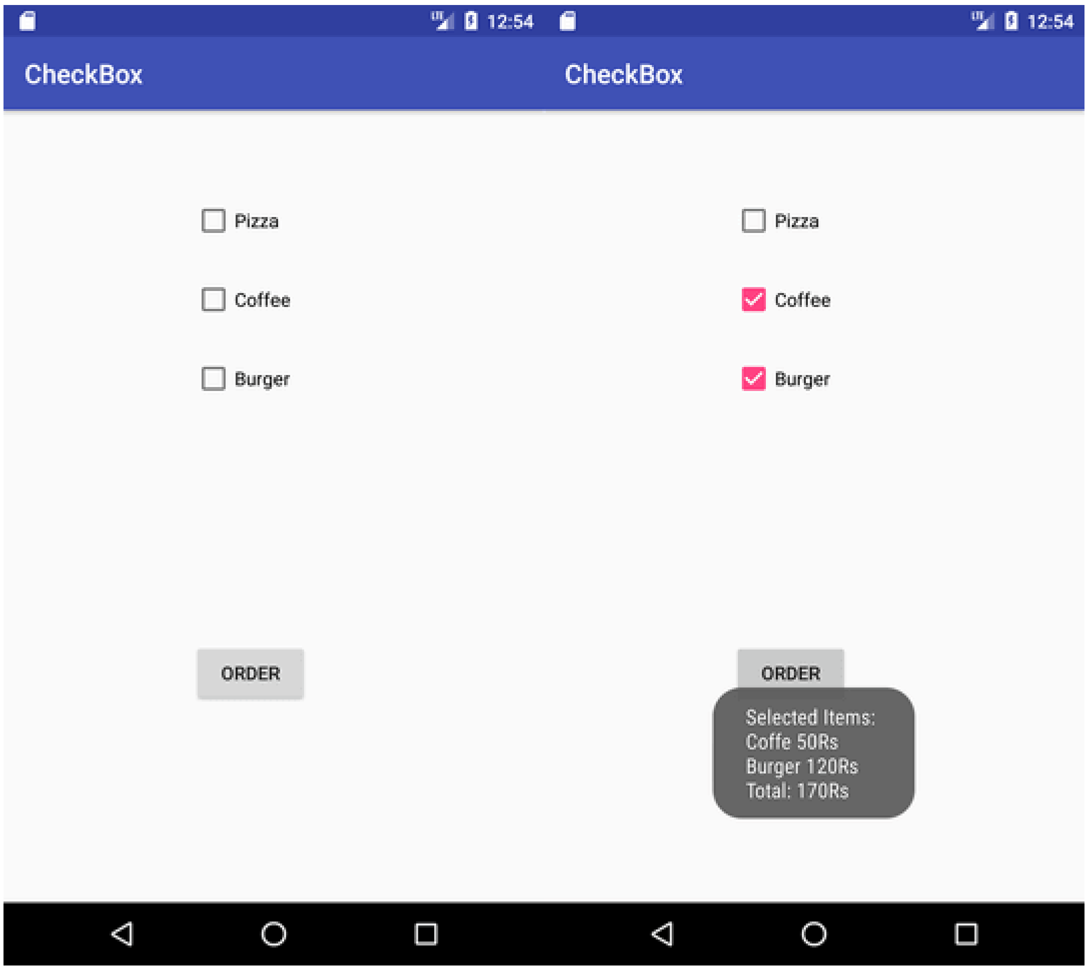
**Android Checkbox Example**

**Android Checkbox:** is a type of two state button either checked or unchecked.
There can be a lot of usage of checkboxes. For example, it can be used to know the hobby of the user, activate/deactivate the specific action etc.

```
package example.javatpoint.com.checkbox;
 import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox pizza,coffe,burger;
    Button buttonOrder;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButtonClick();
```

```
  }
  public void addListenerOnButtonClick(){
     //Getting instance of CheckBoxes and Button from the activty_main.xml file
     pizza=(CheckBox)findViewById(R.id.checkBox);
     coffe=(CheckBox)findViewById(R.id.checkBox2);
     burger=(CheckBox)findViewById(R.id.checkBox3);
     buttonOrder=(Button)findViewById(R.id.button);
       //Applying the Listener on the Button click
     buttonOrder.setOnClickListener(new View.OnClickListener(){
         @Override
        public void onClick(View view) {
           int totalamount=0;
           StringBuilder result=new StringBuilder();
           result.append("Selected Items:");
           if(pizza.isChecked()){
              result.append("\nPizza 100Rs");
              totalamount+=100;
           }
           if(coffe.isChecked()){
              result.append("\nCoffe 50Rs");
              totalamount+=50;
           }
           if(burger.isChecked()){
              result.append("\nBurger 120Rs");
              totalamount+=120;
           }
           result.append("\nTotal: "+totalamount+"Rs");
           //Displaying the message on the toast
           Toast.makeText(getApplicationContext(), result.toString(), Toast.LENGTH_
LONG).show();
        }

     });
  }
}
```

**Output:**



Selected Items:
Coffe 50Rs
Burger 120Rs
Total: 170Rs

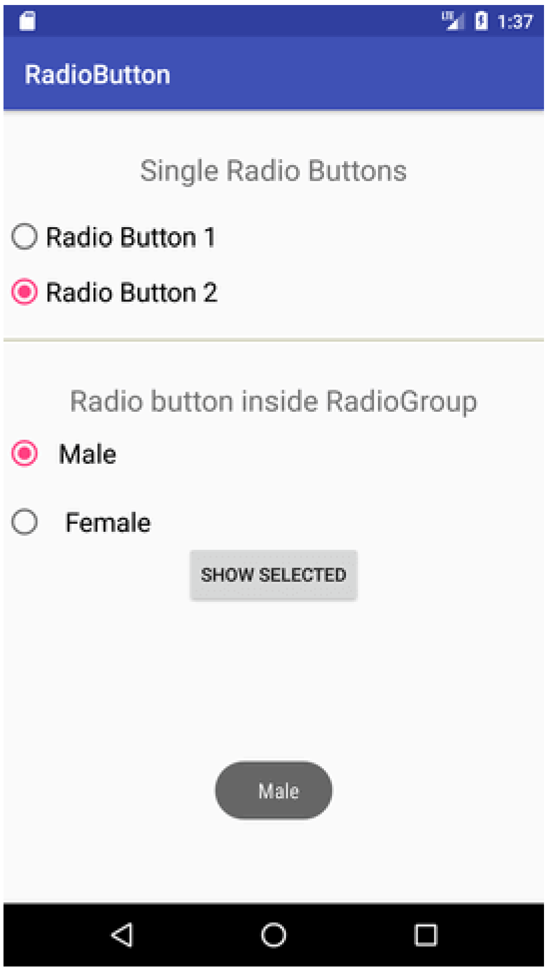
**Android Radio Button**

**Radio Button** is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

Radio Button is generally used with *Radio Group*. Radio Group contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

```
package example.javatpoint.com.radiobutton;
  import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
  public class MainActivity extends AppCompatActivity {
    Button button;
    RadioButton genderradioButton;
    RadioGroup radioGroup;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
    }
    public void onclickbuttonMethod(View v){
        int selectedId = radioGroup.getCheckedRadioButtonId();
        genderradioButton = (RadioButton) findViewById(selectedId);
        if(selectedId==-1){
            Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH_SHORT
).show();
        }
        else{
            Toast.makeText(MainActivity.this,genderradioButton.getText(), Toast.LENGT
H_SHORT).show();
        }

    }
}
```

**Output:**

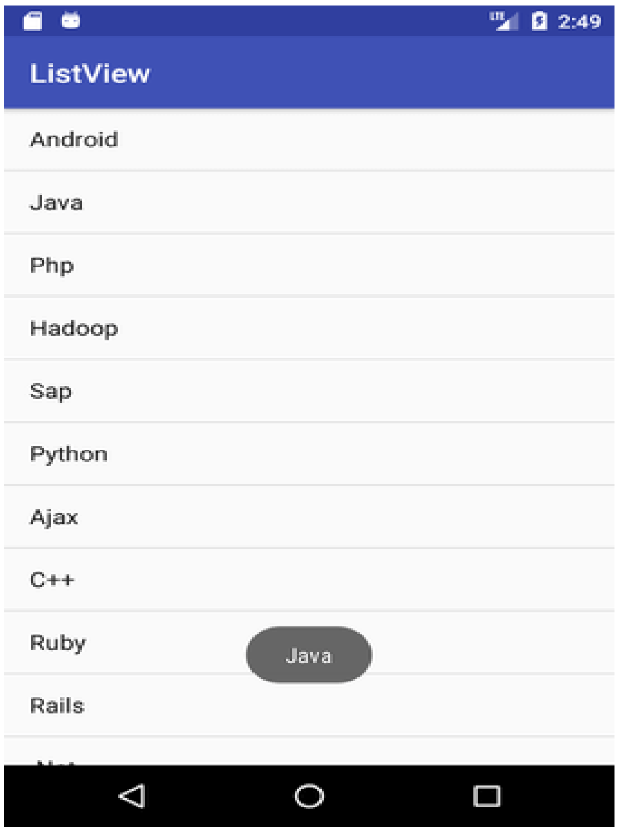


**Android List View**

Android List View is a view which contains the group of items and displays in a scrollable list. List View is implemented by importing *android.widget.ListView* class. List View is a default scrollable which does not use other scroll view.

```
package listview.example.com.listview;
  import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
  public class MainActivity extends AppCompatActivity {
    ListView listView;
    TextView textView;
    String[] listItem;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
          listView=(ListView)findViewById(R.id.listView);
        textView=(TextView)findViewById(R.id.textView);
        listItem = getResources().getStringArray(R.array.array_technology);
        final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, android.R.id.text1, listItem);
        listView.setAdapter(adapter);
                listView.setOnItemClickListener(new AdapterView.OnItemClickL
            istene() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int
position, long l) {
                // TODO Auto-generated method stub
                String value=adapter.getItem(position);
                Toast.makeText(getApplicationContext(),value,Toast.LENGTH_SHO
RT).show();
             }
             }
            );
             }
            }
```

**Output:**



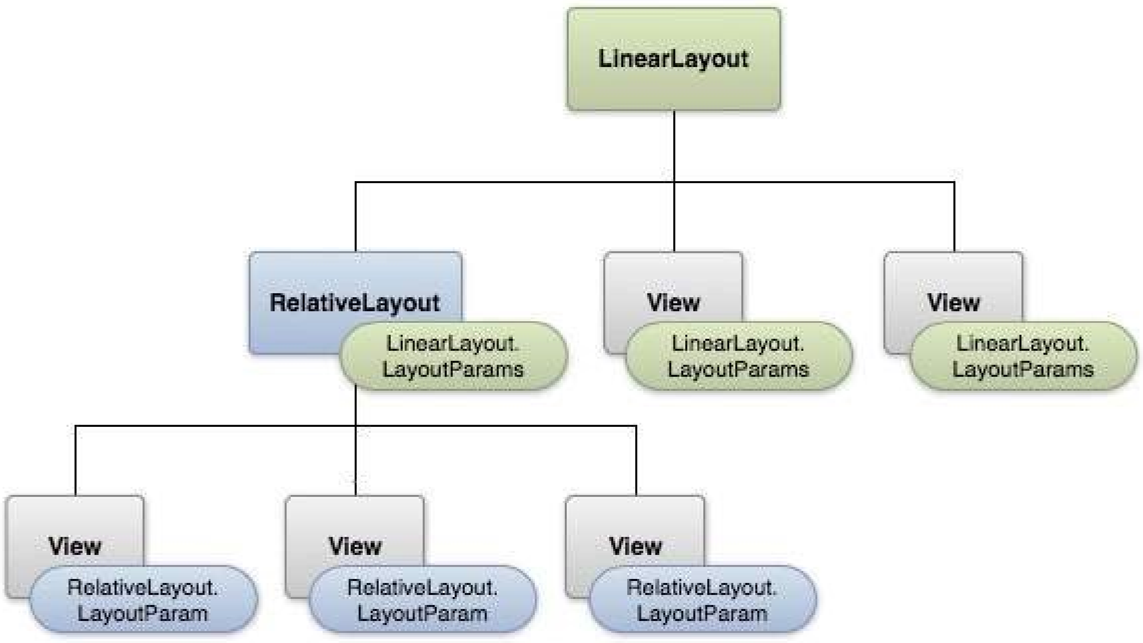
```
ListView
Android
Java
Php
Hadoop
Sap
Python
Ajax
C++
Ruby
Rails
```

## 2.4 View and Layout

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **View Group** is a subclass of **View** and provides invisible container that hold other Views or other View Groups and define their layout properties.

At third level we have different layouts which are subclasses of View Group class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/View Group** objects or you can declare your layout using simple XML file **main_layout.xml** which is in the res/layout folder of your project.

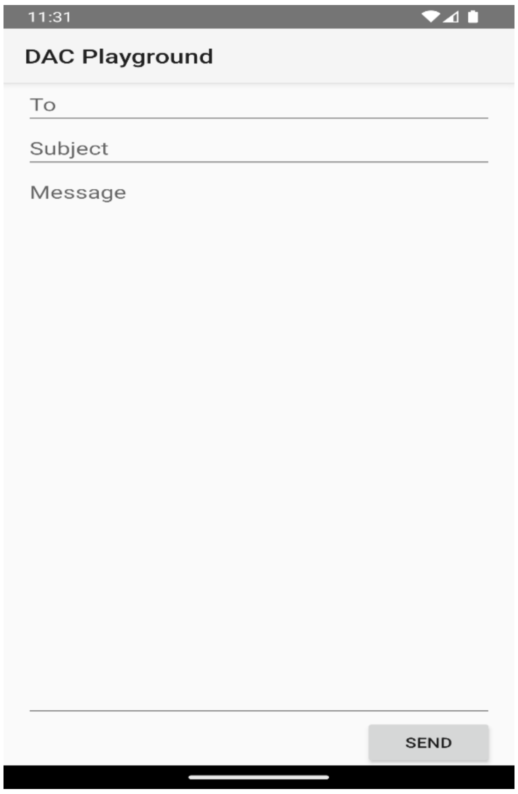

LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

**a. Linear**

Linear Layout is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android: orientation attribute. All children of a Linear Layout are stacked one after the other, so a vertical list only has one child per row, no matter how wide they are. A horizontal list is only one row high, and it's the height of the tallest child, plus padding. A Linear Layout respects margins between children, and the gravity—right, center, or left alignment—of each child.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
```

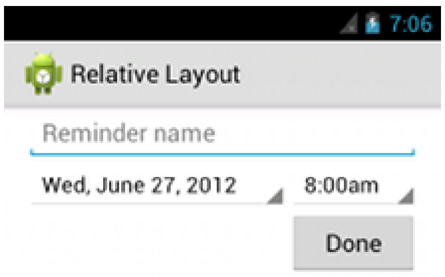
```
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="end"
        android:text="@string/send" />
</LinearLayout>
```



**b. Relative**

Relative Layout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or center).

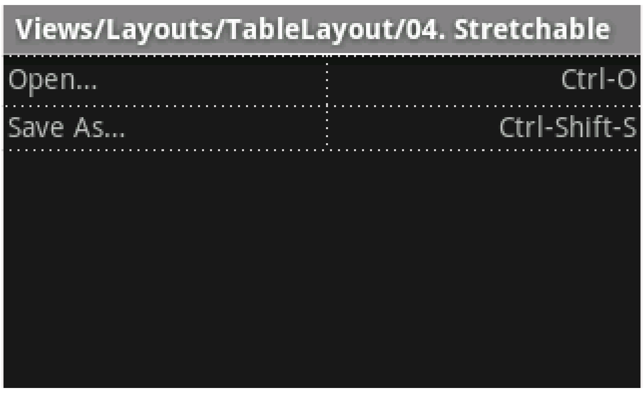
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp" >
   <EditText
      android:id="@+id/name"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/reminder" />
   <Spinner
      android:id="@+id/dates"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/name"
      android:layout_alignParentLeft="true"
      android:layout_toLeftOf="@+id/times" />
   <Spinner
      android:id="@id/times"
      android:layout_width="96dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/name"
      android:layout_alignParentRight="true" />
   <Button
      android:layout_width="96dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/times"
      android:layout_alignParentRight="true"
      android:text="@string/done" />
</RelativeLayout>
```



**c. Table**

Android Table Layout is a View Group subclass that arranges child views into rows and columns. It functions similarly to an HTML table, with the same number of columns as the row with the most cells. Table Layout doesn't display borders between rows, columns, or cells.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:stretchColumns="1">
   <TableRow>
      <TextView
         android:text="@string/table_layout_4_open"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_open_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>

   <TableRow>
      <TextView
         android:text="@string/table_layout_4_save"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_save_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>
</TableLayout>
```



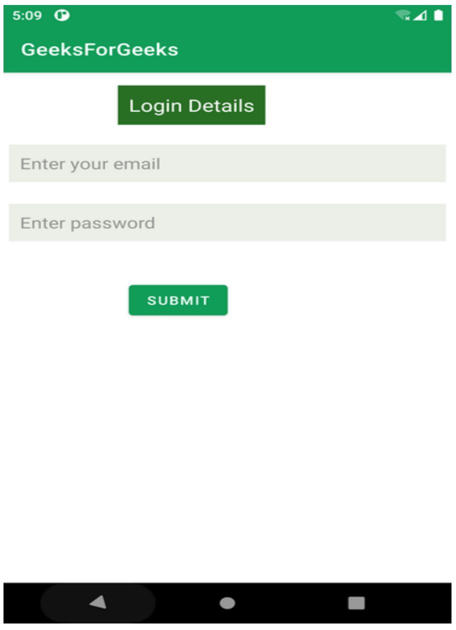
Views/Layouts/TableLayout/04. Stretchable

| | |
|---|---|
| Open... | Ctrl-O |
| Save As... | Ctrl-Shift-S |

**d. Frame**

Android Frame layout is a View Group subclass that is used to specify the position of multiple views placed on top of each other to represent a single view screen. Generally, we can say Frame Layout simply blocks a particular area on the screen to display a single view. Here, all the child views or elements are added in stack format means the most recently added child will be shown on the top of the screen. But we can add multiple children's views and control their positions only by using gravity attributes in Frame Layout.

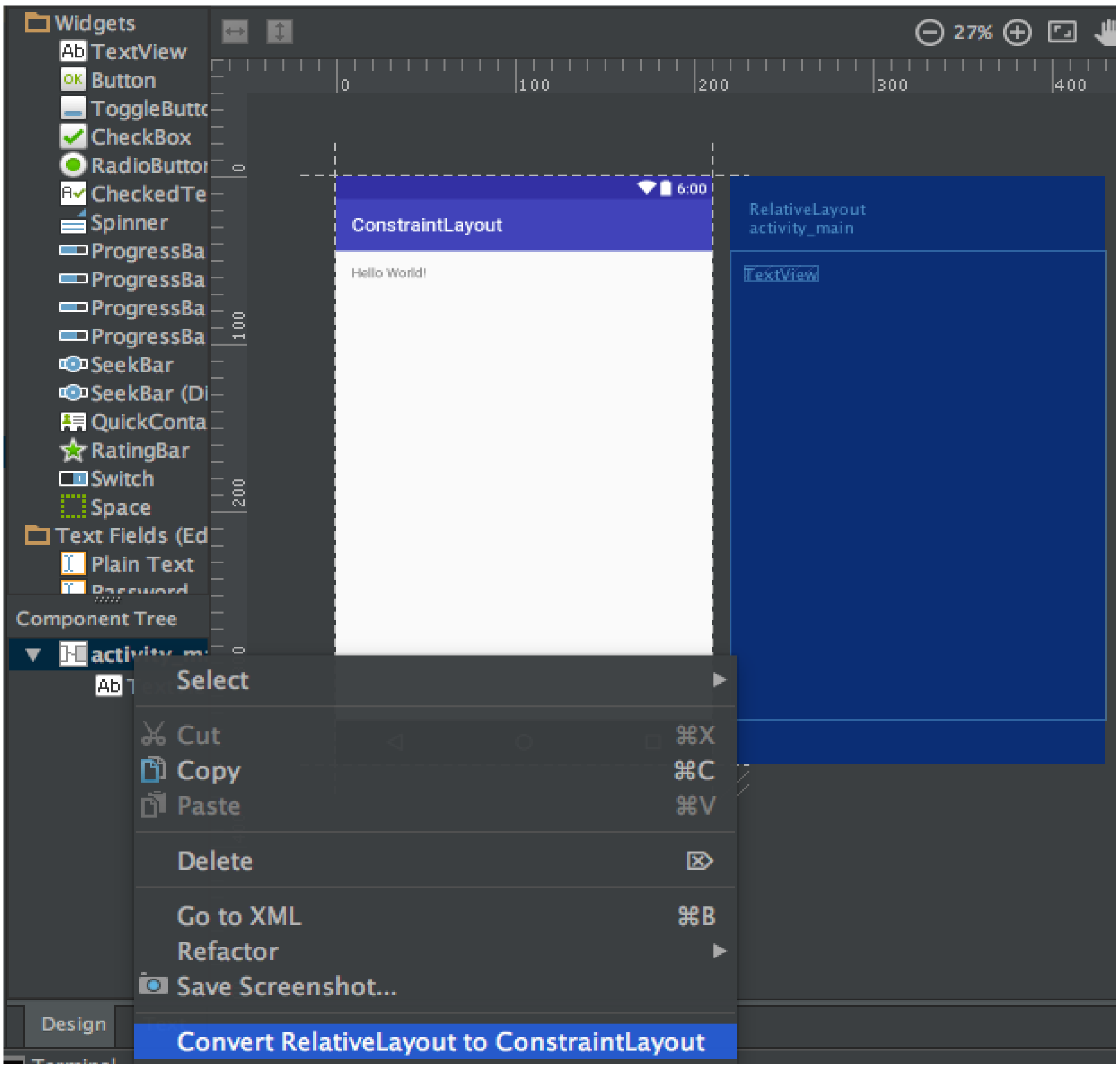
```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.EditText
import android.widget.TextView

class MainActivity : AppCompatActivity() {

  private lateinit var textView: TextView
  private lateinit var editText1: EditText
  private lateinit var editText2: EditText

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    // finding the UI elements

    textView = findViewById(R.id.txtvw1)
    editText1 = findViewById(R.id.editText1);
    editText2 = findViewById(R.id.editText2);
  }}
```



**e. Constraint layout**

Android Constraint Layout is a layout manager that lets you position and size widgets in a flexible way. It's similar to Relative Layout, but it's more powerful and integrates seamlessly with Android Studio's Layout Editor.

To define a layout with Constraint Layout, you assign constraints for each child view or widget relative to other views. A constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. For example, you can constrain a view to both sides on an axis to center it on that axis. You can adjust the bias value for centering using the slider in the Attributes panel under the Layout tab



## Self-Check - 2: Use main building blocks

**Questionnaire**

1. What is the main building block of an Android app?

   **Answer:**

2. What is an Activity in Android?

   **Answer:**

3. How do Activities communicate in Android?

   **Answer:**

4. What is the Activity lifecycle in Android?

   **Answer:**

5. Name some methods in the Activity lifecycle.

   **Answer:**

6. When does onPause() method get called in the Activity lifecycle?

   **Answer:**

7. What are UI widgets in Android?

   **Answer:**

8. How do you handle Button clicks in Android?

   **Answer:**

9. Which widget is used for displaying images in Android?

   **Answer:**

10. What is a View in Android?

    **Answer:**

11. Name different types of layouts in Android.

    **Answer:**

12. How do you define UI components in Android?

    **Answer:**

13. What is the purpose of using XML layout files in Android?

    **Answer**

14. Why is understanding the Activity lifecycle important for Android developers?

    **Answer:**

15. How can you optimize UI performance in Android?

    **Answer:**

## Answer Key - 2: Use main building blocks

1. What is the main building block of an Android app?
   **Answer:** The main building block of an Android app is the

2. What is an Activity in Android?
   **Answer:** An Activity represents a single screen with a user interface that users can interact with.

3. How do Activities communicate in Android?
   **Answer:** Activities can communicate through Intents, which are messages that allow components to request functionality from other components.

4. What is the Activity lifecycle in Android?
   **Answer:** The Activity lifecycle describes the stages an Activity goes through from creation to destruction.

5. Name some methods in the Activity lifecycle.
   **Answer:** onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().

6. When does onPause() method get called in the Activity lifecycle?
   **Answer:** onPause() is called when the Activity is partially obscured by another Activity.

7. What are UI widgets in Android?
   **Answer:** UI widgets are components like TextView, EditText, Button, etc., used to interact with users in the app's interface.

8. How do you handle Button clicks in Android?
   **Answer:** Use onClickListener to handle Button clicks and define actions in the onClick() method.

9. Which widget is used for displaying images in Android?
   **Answer:** ImageView is used to display images in Android applications.

10. What is a View in Android?
    **Answer:** A View is a basic building block for user interface components in Android, representing a rectangular area on the screen.

11. Name different types of layouts in Android.
    **Answer:** LinearLayout, RelativeLayout, ConstraintLayout, FrameLayout, GridLayout.

12. How do you define UI components in Android?
    **Answer:** UI components are defined in XML layout files located in the res/layout/ directory.

13. What is the purpose of using XML layout files in Android?
    **Answer:** XML layout files define the structure and appearance of the app's user interface, allowing for easy modification and separation of code and design.

14. Why is understanding the Activity lifecycle important for Android developers?
    **Answer:** Understanding the Activity lifecycle helps developers manage the state and behavior of their app throughout its lifespan, ensuring a smooth user experience.

15. How can you optimize UI performance in Android?
    **Answer:** Optimize UI performance by using efficient layouts, minimizing nested layouts, recycling views (e.g., with RecyclerView), and reducing unnecessary computations in UI operations.

## Job Sheet 2.1: Use Activity Lifecycle

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Create a new Android project with a single MainActivity.
6. Implement the key methods of the Activity lifecycle (onCreate, onStart, onResume, onPause, onStop, onDestroy).
7. Print log messages in each method to observe the sequence of lifecycle events.
8. Add a button to the layout and handle its click event in the MainActivity
9. Run the app and show your work to the trainer.
10. Clean your work place as per standard procedure.
11. Turn off the computer and clean your workplace.

## Specification Sheet-2.1: Use Activity Lifecycle

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet 2.2: UI Widgets and Layout

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Open Android Studio and click on "Start a new Android Studio project".
6. Choose "Empty Activity" and name the project "MyUIApp".
7. Choose the language as Kotlin and click "Next".
8. Choose the minimum SDK version as API 21 and click "Finish".
9. Design a simple UI using various UI widgets like TextView, Button, EditText, and ImageView.
10. Create a layout (e.g., LinearLayout or RelativeLayout) to arrange these widgets.
11. Customize the appearance of widgets using attributes like text color, background color, etc.
12. Implement functionality to update the TextView content when a button is clicked.
13. Run the App and customize UI
14. Show your work to the trainer.
15. Clean your work place as per standard procedure.
16. Turn off the computer and clean your workplace.

## Specification Sheet-2.2: UI Widgets and Layout

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Job Sheet 2.3: Create view and Layout with following activity

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. Create a new Android project with a blank MainActivity.
6. Define a custom view class (e.g., CustomView) that extends View.
7. Override the onDraw method in the custom view class to draw a simple shape (e.g., a rectangle or circle).
8. In the MainActivity, add an instance of the custom view to the layout and observe the drawn shape..
9. Run the App and Show your work to the trainer.
10. Clean your work place as per standard procedure.
11. Turn off the computer and clean your workplace.

## Specification Sheet-2.3: Create View and Layout

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Learning Outcome 3: Create android user interface

| | |
|---|---|
| Assessment Criteria | 1. Intents are created.<br>2. UI with fragments and action bar is designed.<br>3. A simple activity layout is designed for some basic user operation. |
| Conditions and Resources | • Actual workplace or training environment<br>• CBLM<br>• Handouts<br>• Job related tools, equipment, and materials<br>• Multimedia Projector<br>• Paper, Pen, Pencil, and Eraser<br>• Internet Facilities<br>• Whiteboard and Marker |
| Contents | 1. Create Intents<br>2. Design UI with fragments and action bar<br>3. Simple activity layout |
| Activities/job/Task | 1. Create android user interface |
| Training Methods | • Blended<br>• Discussion<br>• Presentation<br>• Demonstration<br>• Guided Practice<br>• Individual Practice<br>• Project Work<br>• Problem Solving<br>• Brainstorming |
| Assessment Methods | Assessment methods may include but not limited to<br>• Written Test<br>• Demonstration<br>• Oral Questioning |

## Learning Experience 3: Create android user interface

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

| Learning Activities | Recourses/Special Instructions |
|---|---|
| 1. Trainee will ask the instructor about the learning materials | 1. Instructor will provide the learning materials **Create android user interface** |
| 2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Create android user interface" | 2. Read Information sheet 3: Create android user interface<br>3. Answer Self-check 3: Create android user interface<br>4. Check your answer with Answer key 3: Create android user interface |
| 3. Read the Job/Task Sheet and Specification Sheet and perform job/Task | 5. Job/Task Sheet and Specification Sheet<br>**Job Sheet 3.1:** Create android user interface<br>**Specification Sheet-3.1:** Create android user interface |

## Information Sheet 3: Create android user interface

**Learning Objective:** After completion of this information sheet, the learners will be able to explain, define and interpret the following contents

3.1 Create Intents
3.2 Design UI with fragments and action bar
3.3 Simple activity layout Linear

### 3.1 Create Intents

In Android, it is quite usual for users to witness a jump from one application to another as a part of the whole process, for example, searching for a location on the browser and witnessing a direct jump into Google Maps or receiving payment links in Messages Application (SMS) and on clicking jumping to PayPal or GPay (Google Pay). This process of taking users from one application to another is achieved by passing the Intent to the system. Intents, in general, are used for navigating among various activities within the same application, but note, is not limited to one single application, i.e., they can be utilized from moving from one application to another as well.

Intents could be Implicit, for instance, calling intended actions, and explicit as well, such as opening another activity after some operations like onClick or anything else. Below are some applications of Intents:

- Sending the User to Another App
- Getting a Result from an Activity
- Allowing Other Apps to Start Your Activity

**Types of Android Intents**

There are two types of intents in android

a. Implicit
b. Explicit

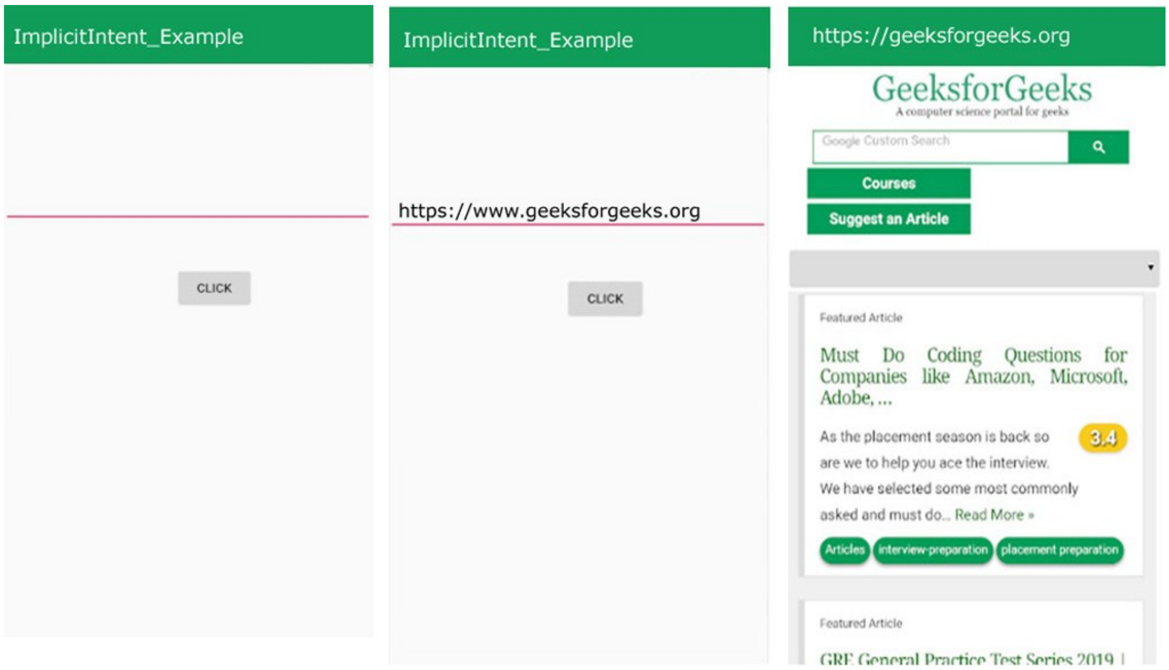
**a. Implicit Intent**

Implicit Intent does not specify the component. In such a case, intent provides information on available components provided by the system that is to be invoked. For example, you may write the following code to view the webpage.

**Syntax**:

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.geeksforgeeks.org/"));
startActivity(intent);
```

**For Example:** In the below images, no component is specified, instead, an action is performed i.e. a webpage is going to be opened. As you type the name of your desired webpage and click on the ‘CLICK’ button. Your webpage is opened.


ImplicitIntent_Example
CLICK
ImplicitIntent_Example
https://www.geeksforgeeks.org
CLICK
https://geeksforgeeks.org
GeeksforGeeks
A computer science portal for geeks
Google Custom Search
Courses
Suggest an Article
Featured Article
Must Do Coding Questions for Companies like Amazon, Microsoft, Adobe, ...
As the placement season is back so are we to help you ace the interview. We have selected some most commonly asked and must do... Read More »
3.4
Articles
interview-preparation
placement preparation
Featured Article
GRE General Practice Test Series 2019

**b. Explicit Intent**

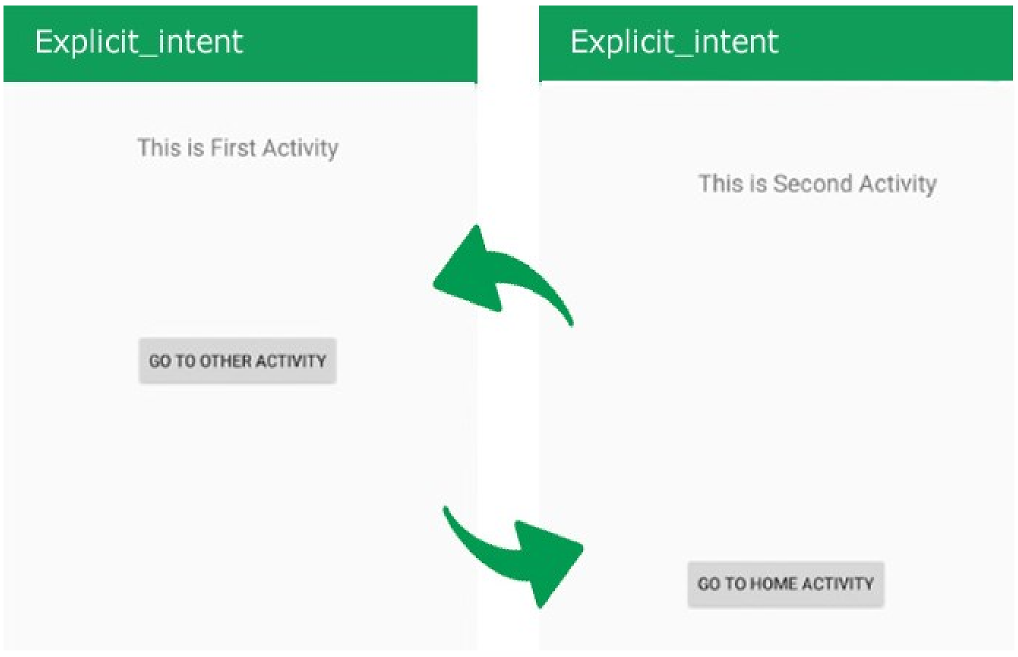
Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.

**Syntax**:

*Intent i = new Intent(getApplicationContext(), ActivityTwo.class);*

*startActivity(i);*

**For Example:** In the below example, there are two activities (FirstActivity, and SecondActivity). When you click on the 'GO TO OTHER ACTIVITY' Button in the FirstActivity, then you move to the SecondActivity. When you click on the 'GO TO HOME ACTIVITY' button in the SecondActivity, then you move to the first activity. This is getting done through Explicit Intent.



## 3.2 Design UI with fragments and action bar

Designing a UI with fragments and an action bar in Android is a great way to create a robust and scalable user interface. Here's a step-by-step guide to help you get started

**Fragments**

Fragments are reusable UI components that can be added or removed dynamically from the activity's UI. They're designed to be modular and reusable, making it easier to manage complex UI layouts. Fragments can contain their own layouts, widgets, and logic, allowing you to create separate, independent sections within your app.

**Action Bar**

The action bar is a component that provides a common navigation and action area at the top of the screen. It typically includes features like:

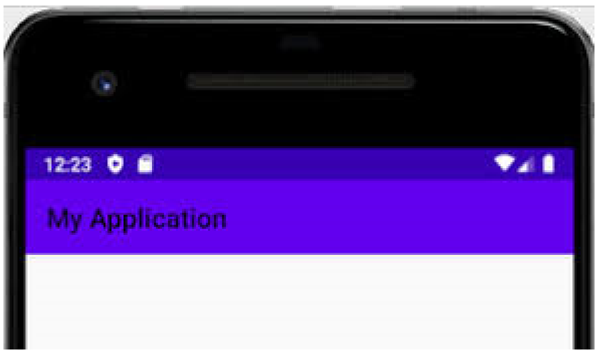
- Navigation icons (e.g., back button, home button)
- Title text
- Action items (e.g., buttons, dropdown menus)

Designing the UI with Fragments and an Action Bar

- Create a new Android project: Start by creating a new Android project in your preferred IDE.
- Define the layout: In your activity's XML file (activity_main.xml), define the overall layout structure using <LinearLayout> or another container view. This will serve as the main container for your fragments.
- Create fragments: Create multiple fragments for different sections of your app (e.g., fragment1.xml, fragment2.xml, ...). Each fragment should contain its own layout and widgets.
- Add fragments to the activity: In your activity's Java/Kotlin code, create instances of each fragment and add them to the activity's layout using FragmentTransaction objects.
- Set up the action bar: In your activity's XML file (activity_main.xml), add an <ActionBar element to define the action bar layout.
- Configure the action bar: In your activity's Java/Kotlin code, set up the action bar by calling getSupportActionBar() and configuring its properties (e.g., title, icon, menu items).
- Implement navigation: Use FragmentTransaction objects to navigate between fragments when users interact with the app (e.g., clicking on a button).
- Style and customize: Customize the appearance of your UI by using themes, styles, and layouts.

**Example Code**

```
<!-- activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <!-- Action bar -->
  <androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar" />

  <!-- Fragment container -->
  <FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

</LinearLayout>
```



## 3.3 Simple activity layout Linear

Creating a simple activity layout using a `LinearLayout` is straightforward and effective for arranging UI components linearly either horizontally or vertically. Here's a step-by-step guide to create such a layout:

**Step-by-Step Guide**

a. Create a New Android Project - Open Android Studio and create a new project with an Empty Activity template.

b. 2. Navigate to Layout XML File - Open the `activity_main.xml` file located in the `res/layout` directory.

c. Define the LinearLayout: -Replace the existing `<RelativeLayout>` or `<ConstraintLayout>` with a `<LinearLayout>` and set its orientation attribute to either `vertical` or `horizontal` based on your layout needs.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp">

   <!-- Add your UI components here -->

</LinearLayout>
```

d. In this example, the `LinearLayout` is set to `vertical` orientation and has `match_parent` width and height.

e. Add UI Components: - Inside the `LinearLayout`, add various UI components like `TextView`, `EditText`, `Button`, etc., using appropriate attributes to control their appearance and behavior.

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Name:" />

<EditText
   android:id="@+id/editTextName"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Enter your name" />

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Submit"
   android:onClick="submitForm" />
```

f. - Adjust `layout_width`, `layout_height`, and other attributes as needed for each component.

g. Handle UI Events (Optional): - Optionally, you can define methods like `submitForm()` in your activity to handle UI events, such as button clicks.

h. Preview Layout:- Switch to the Design tab in Android Studio to preview the layout visually. Ensure the components are arranged as intended.
i. Run Your App: - Run your app on an emulator or physical device to see how the layout appears and functions.

Example Layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="16dp">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Name:" />

  <EditText
    android:id="@+id/editTextName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter your name" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Submit"
    android:onClick="submitForm" />
</LinearLayout>
```

In this layout:
- A `TextView` displays a label "Name:".
- An `EditText` allows the user to input their name.
- A `Button` triggers a form submission action.

Adjust the attributes and components based on your specific application requirements. This simple `LinearLayout` layout provides a basic structure for organizing UI elements in an Android app.

## Self-Check - 3: Create android user interface

**Questionnaire**

1. What is the main goal of Android user interface (UI) design?

   **Answer:**

2. Name two essential UI components in Android.

   **Answer:**

3. How can you define a UI layout in Android?

   **Answer:**

4. What is an Intent in Android?

   **Answer:**

5. How do you create an explicit Intent to start another Activity?

   **Answer:**

6. What are implicit Intents used for?

   **Answer:**

7. Why use Fragments in Android UI design?

   **Answer:**

8. What is the Action Bar used for in Android?

   **Answer:**

9. How do you add Fragments to an Activity?

   **Answer:**

10. What is a LinearLayout in Android layouts?

    **Answer:**

11. What attributes does LinearLayout use to define orientation?

    **Answer:**

12. Name one advantage of using LinearLayout.

    **Answer:**

13. How do you create a TextView in XML?

    **Answer:**

14. What attribute is used to handle button clicks in XML?

    **Answer:**

# Answer Key - 3: Create android user interface

1. What is the main goal of Android user interface (UI) design?
   **Answer:** The main goal is to create intuitive and visually appealing interfaces that enhance user experience.
2. Name two essential UI components in Android.
   **Answer:** TextView and Button are essential components for displaying text and triggering actions, respectively.
3. How can you define a UI layout in Android?
   **Answer:** UI layouts are defined using XML files located in the res/layout directory, specifying components and their attributes.
4. What is an Intent in Android?
   **Answer:** An Intent is a messaging object used to request an action from another app component or system.
5. How do you create an explicit Intent to start another Activity?
   **Answer:** Use Intent intent = new Intent(this, TargetActivity.class); where TargetActivity is the class of the activity you want to start.
6. What are implicit Intents used for?
   **Answer:** Implicit Intents are used to activate components without specifying the exact component to be activated, typically to request an action from another app.
7. Why use Fragments in Android UI design?
   **Answer:** Fragments enable modularization of UI components, making it easier to create flexible layouts that adapt to different screen sizes.
8. What is the Action Bar used for in Android?
   **Answer:** The Action Bar provides navigation, actions, and branding elements at the top of an activity's window.
9. How do you add Fragments to an Activity?
   **Answer:** Fragments are added to an activity's layout using <fragment> tags in XML or programmatically using FragmentTransaction.
10. What is a LinearLayout in Android layouts?
    **Answer:** LinearLayout arranges child views in a single direction either vertically or horizontally.
11. What attributes does LinearLayout use to define orientation?
    **Answer:** Use android:orientation="vertical" or android:orientation="horizontal" to specify the layout direction.
12. Name one advantage of using LinearLayout.
    **Answer:** It is simple to use and straightforward for arranging UI components in a linear fashion.
13. How do you create a TextView in XML?
    **Answer:** <TextView android:layout_width="wrap_content" android:layout_height="wrap_content" android:text="Hello World!" />
14. What attribute is used to handle button clicks in XML?
    **Answer:** Use android:onClick="methodName" to specify a method in the activity that handles the button click event.

## Job Sheet 3.1: Create android user interface

**Working Procedure/ Steps:**

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Collect the resources from your trainer as per the job requirement.
5. **Create a new Android project**
   a. Open Android Studio and click on "Start a new Android Studio project".
   b. Choose "Empty Activity" and click "Next".
   c. Fill in the project details:
      i. Project name: e.g., "MyFirstApp"
      ii. Package name: e.g., "com.example.myfirstapp"
      iii. Save location: choose a location on your computer
      iv. Language: Kotlin
      v. Minimum SDK: API 21 (or higher)
   d. Click "Finish" to create the project.
6. **Design the user interface**
   a. Open the activity_main.xml file in the res/layout directory.
   b. Drag and drop UI components from the Palette panel onto the design canvas:
      i. Button
      ii. Text View
      iii. Edit Text
   c. Arrange the components as desired.
   d. Configure the properties of each component:
      i. Button: text="Click me!"
      ii. Text View: text="Hello, World!"
      iii. Edit Text: hint="Enter your name"
7. **Write Kotlin code for the activity**
8. **Run the app**
   a. Connect a physical device or emulator to your computer.
   b. Click the "Run" button or press Shift+F10.

    c. The app will run on the device or emulator, and you should see the UI with a button and edit text.

9. **Step 5: Test the app**
    a. Enter some text in the edit text field.
    b. Click the button.
    c. Verify that the text is displayed in the text view with a greeting message.
10. Run the App and Show your work to the trainer.
11. Clean your work place as per standard procedure.
12. Turn off the computer and clean your workplace.

## Specification Sheet-3.1: Create android user interface

**Conditions for the job:** Work must be carried out in a safe manner and according to relevant competency standards.

**List of required PPE**

| S/N | Name of PPE | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Ergonomic Chair | Wood and foam | No | 1 |
| 02 | Eye protective glass | Metal and Glass | No | 1 |
| 03 | Mask | Surgical mask | 1 | 1 |

**List of required Software**

| S/N | Name of Materials | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | IntelliJ IDEA | Latest version | … | 1 |
| 02 | Android Studio | Latest version | … | 1 |
| 03 | Java | JDK Latest version | … | 1 |

**List of required Tools & Equipment's**

| S/N | Name | Specification | Unit | Required Quantity |
|---|---|---|---|---|
| 01 | Personal Computer or Laptop | Minimum Core i5 7th gen Processor | No | 1 |
| 04 | Internet connection |  | … | 1 |

## Review of Competency

Below is yourself assessment rating for module "Working with Android Basic"

| Assessment of performance Criteria | Yes | No |
|---|---|---|
| History of Apps Development is explained. | | |
| Overview of Android Apps development is described. | | |
| Computer is configured for setting up android application development environment. | | |
| Android Apps Development Environment in a computer is explained. | | |
| Main building block of Android apps is used. | | |
| Activity life cycle is used. | | |
| UI Widgets are used. | | |
| View and Layout are created | | |
| Intents are created. | | |
| UI with fragments and action bar is designed. | | |
| A simple activity layout is designed for some basic user operation. | | |

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

## Development of CBLM

The Competency based Learning Material (CBLM) of '**Working with Android Basic' (Occupation: Android Mobile Application Development, Level-4)** for National Skills Certificate is developed by NSDA with the assistance of SIMEC System Ltd., ECF Consultancy & SIMEC Institute of Technology JV (Joint Venture Firm) in the month of July, 2024 under the contract number of package SD-9B dated 15th January 2024.

| SL No. | Name & Address | Designation | Contact Number |
|---|---|---|---|
| 1 | Engr. Md. Zuwel Parves | Writer | 01737-278906 |
| 2 | Md. Abdul Al Hossain | Editor | 01778-926438 |
| 3 | Engr Md. Zuwel Parves | Co-Ordinator | 01737-278906 |
| 4 | Md. Abdur Razzaque | Reviewer | 01713-304824 |

# Reference

1. https://reflectoring.io/kotlin-design-patterns/
2. https://www.w3schools.com/kotlin/index.php
3. https://www.tutorialspoint.com/kotlin/index.htm
4. https://medium.com/@oriohac/beginner-friendly-implementation-of-the-http-methods-get-post-put-and-delete-from-apis-in-b4e93952aa29
5. https://www.freecodecamp.org/news/http-request-methods-explained/
6. https://code.tutsplus.com/android-from-scratch-using-rest-apis--cms-27117t
7. https://developer.android.com/games/sdk/performance-tuner/custom-engine/enable-api
8. https://www.geeksforgeeks.org/networking-and-api-integration-in-android/
9. https://firebase.google.com/docs/database/android/start
10. https://www.javatpoint.com/firebase-types-of-message