

Mobile application development

Mobile Technologies – Definition, Types, Uses, Advantages

- A mobile application (also called a mobile app) is a type of application designed to run on a mobile device, which can be a smartphone or tablet computer. Even if apps are usually small software units with limited function, they still manage to provide users with quality services and experiences.
- In which a user utilizes a mobile phone to perform communications-related tasks, such as communicating with friends, relatives, and others.
- It is used to send data from one system to another. Portable two-way communications systems, computing devices, and accompanying networking equipment make up mobile technology.
- Mobile technology is largely employed in cellular communication systems and other related areas.
- It employs a network architecture that allows multiple transmitters to deliver data on a single channel at the same time.
- Mobile technology has progressed from a simple phone and texting device to a multi-tasking system that can be used for GPS navigation, internet browsing, gaming, and instant messaging, among other things.
- The future of computer technology is dependent on wireless networking and mobile computing.
- Through tablets and small PCs, mobile technology is becoming increasingly popular. This smartphone system has since been improved to a big multitasking computer that can be used for GPS navigation, gaming, internet browsing, and instant messaging.
- Tablets and portable laptops have increased the adoption of mobile technology. The mobile networks that connect these devices are referred to as wireless systems. They allow speech, data, and (mobile) apps to be shared between mobile devices.
- Apps are divided into two broad categories:
native apps and web apps.

There are several types of apps currently available.

- **Gaming apps:** The equivalent of computer video games, they are among the most popular types of apps.
- **Productivity apps:** These focus on improving business efficiency by easing various tasks such as sending emails, tracking work progress, booking hotels, and much more.
- **Lifestyle and entertainment apps:** Increasingly popular, these encompass many aspects of personal lifestyle and socialization such as communicating

on social media, as well as sharing (and watching) videos. Some of the most widely known apps such as Netflix, Facebook fall into this category.

Types of Mobile Technologies

Followings are the few famous mobile technologies:

- SMS
- MMS
- 1G (First Generation)
- 2G (Second Generation)
- 3G (Third Generation)
- 4G (fourth generation)
- GSM
- CDMA
- Wi-Fi

Let discuss them one by one in detail:

1. SMS:

- “SMS” stands for “Short Message Service.” It is now the most widely used and oldest text messaging service.
- SMS is a low-cost messaging medium.
- Every text message delivered to a cell phone has become known as SMS. Messages can usually be up to 140 characters long.
- SMS was originally developed for GSM phones, although it is now supported by all major cellular phone networks.
- Although SMS is most commonly used for text messaging between friends or coworkers, it also has a variety of additional uses.
- For example, SMS subscription services can send weather, news, sports updates, and financial quotes to consumers’ phones. Employees may also be notified of sales requests, service stops, and other business-related information via SMS.
- Fortunately, text messages sent via SMS do not require the receiver’s phone to be turned on in order for the message to be delivered. The message will be kept in the SMS service until the receiver switches on his or her phone, at which point it will be transmitted to the recipient’s phone

2. MMS:

- MMS (Multimedia Messaging Service) messaging is a standard method of delivering multimedia material, including messages.
- MMS, as opposed to SMS, can send up to forty seconds of video, one picture, a multi-image slideshow, or audio.
- MMS capability is typically embedded within the text message interface and is turned on automatically when needed.
- If you enter in a text-only message, for example, it will be transmitted by SMS.

- If you include a graphic or video, the multimedia part will be sent via MMS. Similarly, if someone sends you a multimedia message, your phone will automatically receive the file via MMS.
- An MMS message can convey rich media content to mobile devices at any time and from any location.
- MMS texts are packed with photographs and videos, As well as other audios.

3) 1G(First Generation):

1982: First Generation (1G — analog voice only) systems with large heavy phones and poor network quality were introduced.

4) 2G(Second Generation):

- Text messaging was made possible by the second-generation network
- 1992: Second Generation (2G) was deployed with improvements in signaling and hardware that were primarily aimed toward the voice market but, unlike the first-generation systems, used digital modulation to enhance call quality and enable new applications such as Short Messaging Services (SMS) and other low-data-rate (9.6 to 237 kbps) wireless applications.

5) 3G (Third Generation):

- 3G stands for third-generation access technology, which allows mobile phones to connect to the internet.
- Every new technology introduces new frequency bands and data transmission rates.
- The development of 3G connection-based networks in 2001 marked the start of mainstream Internet use on mobile phones.
- Soon after, smartphones were introduced, bringing all of the capabilities of a device into the palm of your hand.
- The signals are transmitted by a network of telephone towers. The user's mobile phone is receiving data from the tower nearest to it.
- 3G technology was revolutionary at the time it was introduced.
- 3G systems are intended for digital phones with a full-screen display and better connectivity.

6) 4G:

- The fourth generation of mobile networking technology is known as 4G, which comes after the 2G and 3G networks.
- Although it's commonly referred to as 4G LTE.
- Most mobile network service providers use it now since it is the most developed technology.
- When it initially came out, 4G revolutionized how we use the mobile internet.
- 4G network connectivity allowed consumers to browse the internet and watch HD films on their mobile devices, thereby turning smartphones into laptops.
- However, as you may have heard, 5G is becoming operational alongside current 3G and 4G mobile networks.

- Most tasks that you can do on a laptop or desktop computer can now be done on mobile devices such as smartphones or tablets. No matter how much data you require, 4 G networks allow you to keep consistent speeds practically anywhere. 4G was launched in the United Kingdom in 2012.
- Expect this to alter in the coming years as 4G contracts become more affordable and 4G network coverage increases across the UK.
- Download speed and Uploading data is also significantly faster.

5. Global System for Mobile technology:

- The (GSM) is an acronym for Global System for Mobile Communication.
- GSM is a cellular technology that is open and digital and is used for mobile communication.

6. Code Division Multiple Access:

- The (CDMA) is an acronym for code division multiple access.
- It is a channel access mechanism that also serves as an example of multiple access. Multiple access simply means that data from multiple transmitters can be delivered onto a single communication channel at the same time.

7. Wi-Fi (Wireless Fidelity):

- Wi-Fi is a wireless networking technology that allows us to connect to a network or to other computers or mobile devices across a wireless channel.
- wireless network that functions as a Local Area Network without the use of cables or other types of cabling.

Use of Mobile technology

- The incorporation of mobile technology into business has aided telecollaboration. Now, people could connect from anywhere using mobile technology, and access the papers and documents they need to complete collaborative work.
- Work is being redefined by mobile technologies. Employees are no longer confined to their desks; they can work from anywhere in the world.
- Mobile technology can help your company save time and money. Employees who work from home save thousands on a regular basis. Mobile phones eliminate the need for costly technology like landline carrier services. Cloud-based services are less expensive than traditional systems. Technology can also help your company become more flexible and productive.
- Mobile technology has the potential to boost productivity significantly. Mobile application integration saves an average of 7.5 hours per week per employee. Workers can also become more productive with the use of smartphones and mobile gadgets.
- The popularity of cloud-based services has skyrocketed in recent years. Cloud-based mobile technology applications have been seen to be more useful than any smartphone, particularly in terms of available storage space.

Advantages of Mobile technology

- **Greater reach and visibility**
 - Mobile apps can be downloaded from app stores like the App Store or Google Play, meaning you can reach a wider audience around the world. Additionally, apps can be easily shared by users, further increasing their reach and visibility.
- **Better user experience**
 - Mobile apps are typically designed specifically for mobile devices and offer an optimized user experience compared to mobile websites. Apps typically have an intuitive user interface and faster navigation, which can improve user satisfaction.
- **Take advantage of the device's features**
 - Mobile applications can take advantage of the features of mobile devices such as camera, GPS, microphone, accelerometer, etc. This allows applications to be more interactive and personalized for users.
- **Improve customer loyalty**
 - Mobile apps allow businesses to connect directly with their customers and offer a personalized and relevant experience. Push notifications can be used to send important information and promotions to users, which can help improve customer loyalty.
- **Generate income**
 - Mobile apps can generate revenue in a variety of ways, such as by selling products or services within the app, by advertising, or by subscribing users. Apps can also help reduce marketing and advertising costs by promoting the brand and products effectively.

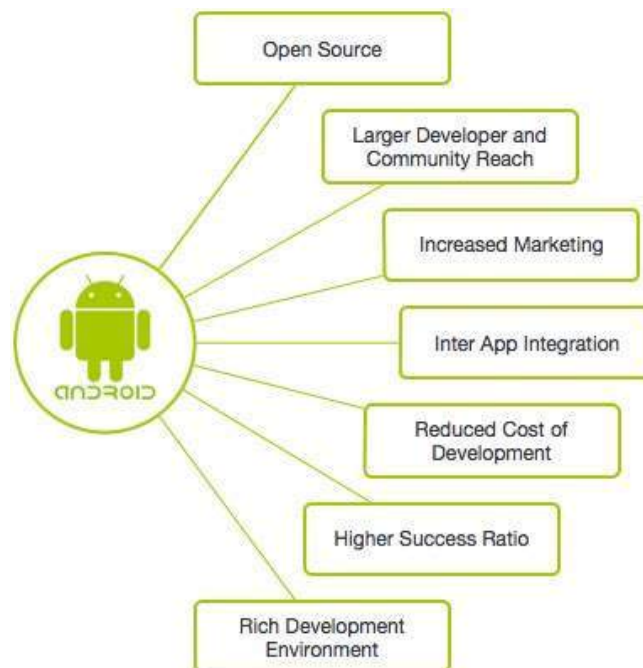
Disadvantages of Mobile technology

- **Higher development cost**
 - Mobile app development can be more expensive than web app development, as specific versions are required to be created for different operating systems (iOS and Android) and devices.
- **Longer development time**
 - The development process for a mobile app can be longer than that of a web app due to the need to create multiple versions for different operating systems and devices, which can delay time to market.
- **Lower audience reaches**
 - Mobile apps are limited by the number of devices they can be downloaded on, while web apps can be accessible from any device with an Internet connection. Additionally, mobile applications may require users to have a stable Internet connection and a sufficient amount of available storage on their devices to download the application.
- **Updates and maintenance**
 - Mobile apps require regular updates to ensure they work correctly on users' devices, and this can be expensive and time-consuming. Additionally, additional testing may be required to ensure that the app works correctly on different operating systems and devices.

Introducing Android

- Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.
- Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

Why Android ?



Android - Application Components

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

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

Components & Description

Activities

- 1 They dictate the UI and handle the user interaction to the smart phone screen.

Services

- 2 They handle background processing associated with an application.

Broadcast Receivers

- 3 They handle communication between Android OS and applications.

Content Providers

- 4 They handle data and database management issues.

Activities

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

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

```
public class MainActivity extends Activity {  
}
```

Services

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

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

```
public class MyService extends Service {  
}
```

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system.
- For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device

and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

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

```
public class MyReceiver extends BroadcastReceiver {  
    public void onReceive(context,intent){}  
}
```

Content Providers

- A content provider component supplies data from one application to others on request.
- Such requests are handled by the methods of the *ContentResolver* class.
- The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider {  
    public void onCreate(){  
    }  
}
```

Additional Components

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

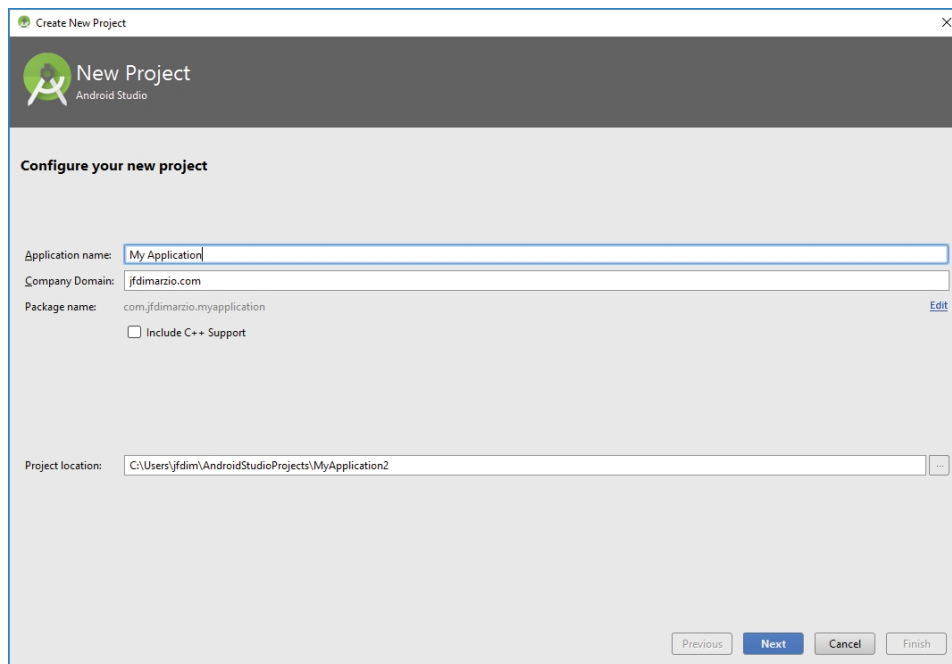
- | o | Components & Description |
|---|--|
| 1 | Fragments
Represents a portion of user interface in an Activity. |
| 2 | Views
UI elements that are drawn on-screen including buttons, lists forms etc. |
| 3 | Layouts
View hierarchies that control screen format and appearance of the views. |
| 4 | Intents
Messages wiring components together. |
| 5 | Resources
External elements, such as strings, constants and drawable pictures. |
| 6 | Manifest
Configuration file for the application. |

Exploring the development environment

Explore the Android Studio Integrated Development Environment, which is also known as the IDE. Basically, the IDE is the interface between you and Android Studio. The more you know about the tools, windows, and options that are available to you in Android Studio, the faster you will be able to produce code and the more confident you will be at creating applications.

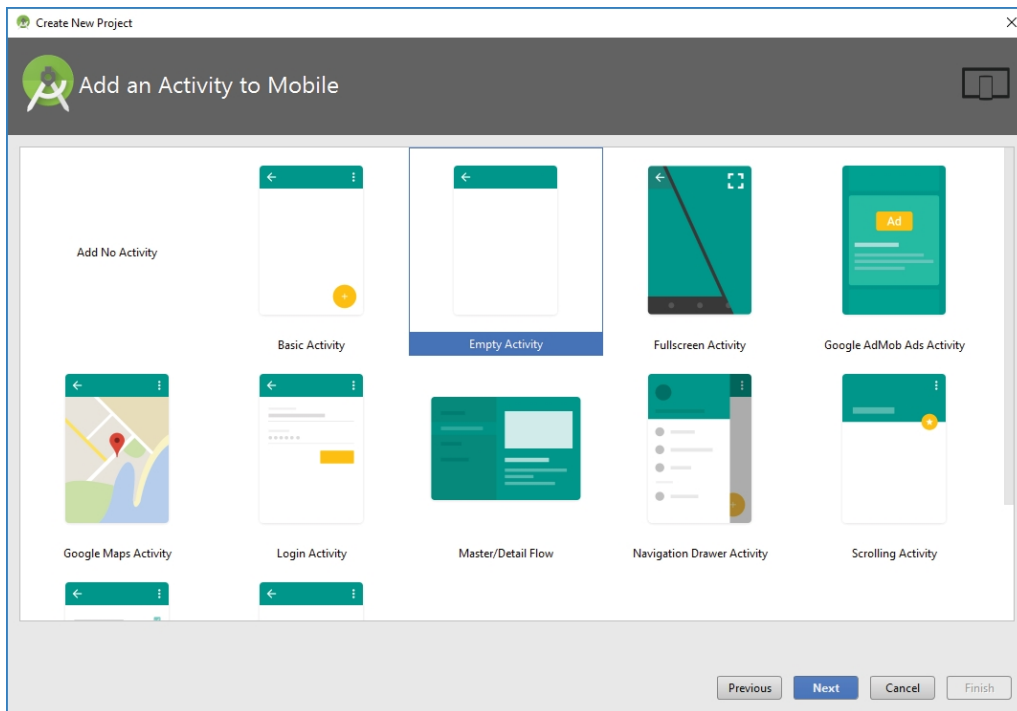
1. The Android Studio welcome screen contains an option for you to open existing projects that you might have already created in Android Studio. It also presents options for opening a project from VCS, and importing projects from other IDEs, such as Eclipse.

2. Click the Start a New Android Studio Project option from the Android Studio welcome screen. You should now see the Create New Project screen, in which enables you to configure some of the basic options for your project.

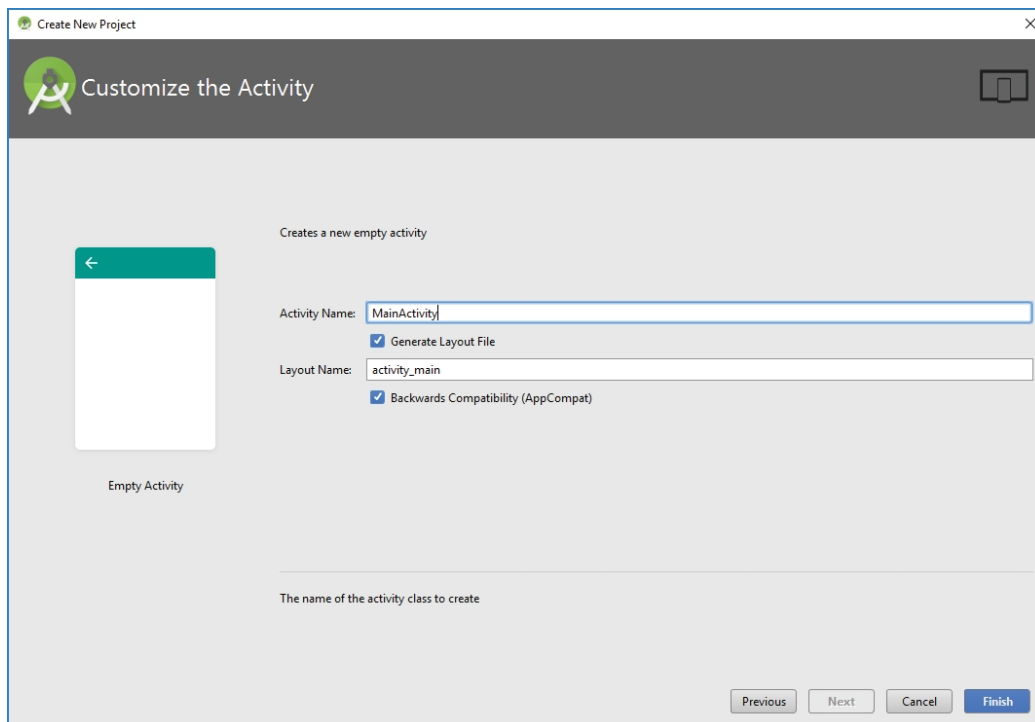
The screenshot shows the 'Create New Project' dialog box in Android Studio. The title bar says 'Create New Project'. The main header area has the Android Studio logo and the text 'New Project' and 'Android Studio'. Below this, the section 'Configure your new project' contains several input fields: 'Application name' with the value 'My Application', 'Company Domain' with 'jfdimarzio.com', and 'Package name' with 'com.jfdimarzio.myapplication'. There is an 'Edit' link next to the package name. Below these is an unchecked checkbox labeled 'Include C++ Support'. At the bottom, the 'Project location' field shows the path 'C:\Users\jfdim\AndroidStudioProjects\MyApplication2' with a browse button (three dots). At the very bottom are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Finish'.

- The final option on the Create New Project screen is the path to which Android Studio will save your new project. I typically accept the default here because it makes it easier for me to find projects in the future. Click Next to continue.
- The next screen allows you to select the form factor on which your application will run.
- The final option on the Create New Project screen is the path to which Android Studio will save your new project.
- The options on this screen range from Add No Activity to Tabbed Activity. For example, if you were to select the Google Maps Activity

option, Android Studio would create for you a project with a basic activity that contains a Google Map in it already.



- The default option is Empty Activity. This is the most useful for our examples because it creates a basic activity for you, with no code in it.
- Click Next to go to the Customize the Activity screen



The Customize the Activity screen contains two options, one for naming your activity, and one for Naming the main layout.

Click the Finish button to finish creating the project and jump into exploring the IDE.

MENU BARS And RUN & DEBUG:

- The upper portion of the IDE represents the menu bars or ribbons. Here, as with most applications that you have used in the past, you have all of your options for interacting directly with the IDE.
- The most important ones to note are the green arrow, which represents the Run app option, and the green arrow with a bug behind it, which is the Debug App option.

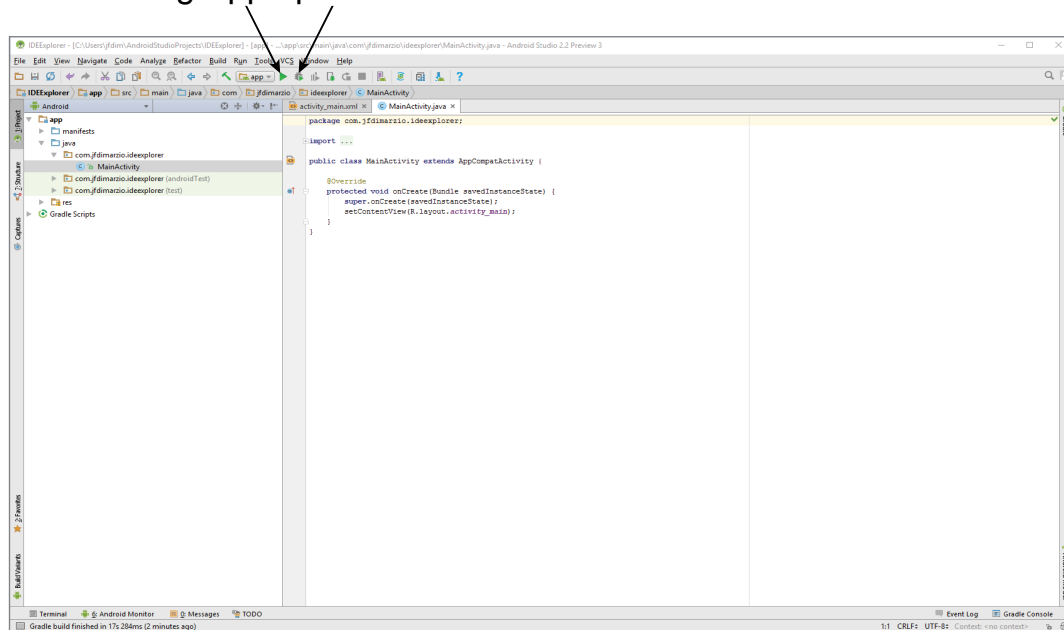
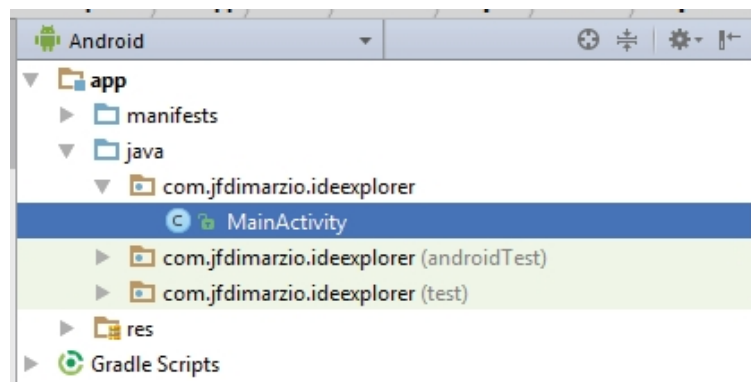


Figure 2-6

The left side of the IDE shows the Project window, The Project window enables you to quickly navigate the files within your project. By default, the Project window is set to the Android view (seen just above the Project window display). To change the view, click the word Android and use the drop-down list of options to make the change.

Project Window/File Structure: -



- **AndroidManifest.xml:** Every project in Android includes a manifest file, which is AndroidManifest.xml, stored in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the Activities, Services, Content Providers and Broadcast Receiver that make the application and using Intent Filters and Permissions, determines how they co-ordinate with each other and other applications.

- **java:** The Java folder contains the Java source code files. These files are used as a controller for controlled UI (Layout file). It gets the data from the Layout file and after processing that data output will be shown in the UI layout. It works on the backend of an Android application.

- **drawable:** A Drawable folder contains resource type file (something that can be drawn). Drawable may take a variety of file like Bitmap (PNG, JPEG), Nine Patch, Vector (XML), Shape, Layers, States, Levels, and Scale.

- **layout:** A layout defines the visual structure for a user interface, such as the UI for an Android application. This folder stores Layout files that are written in XML language. You can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout file.

- **mipmap:** Mipmap folder contains the Image Asset file that can be used in Android Studio application. You can generate the following icon types like Launcher icons, Action bar and tab icons, and Notification icons.

- **colors.xml:** colors.xml file contains color resources of the Android application. Different color values are identified by a unique name that can be used in the Android application program. Below is a sample colors.xml file:

```
<?xml version="1.0" encoding="utf-8"?> <resources>
<color name=" colorPrimary">#3F51B5</color> <color
name=" colorPrimaryDark">#303F9F</color> <color
name="
colorAccent">#FF4081</color> </resources>
```

- **strings.xml:** The strings.xml file contains string resources of the Android application. The different string value is identified by a unique name that can be used in the Android application program. This file also stores string array by using XML language. Below is a sample colors.xml file .

```
<resources>
<string name="app_name">
Text</string> </resources>
```

- **styles.xml:** The styles.xml file contains resources of the theme style in the Android application. This file is written in XML language. Below is a sample styles.xml file:

```
<resources>
<!-- -- Base application theme. > <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar"> <!-- --
Customize your theme here. >
<item name="colorPrimary">@color/colorPrimary</item>
```

```

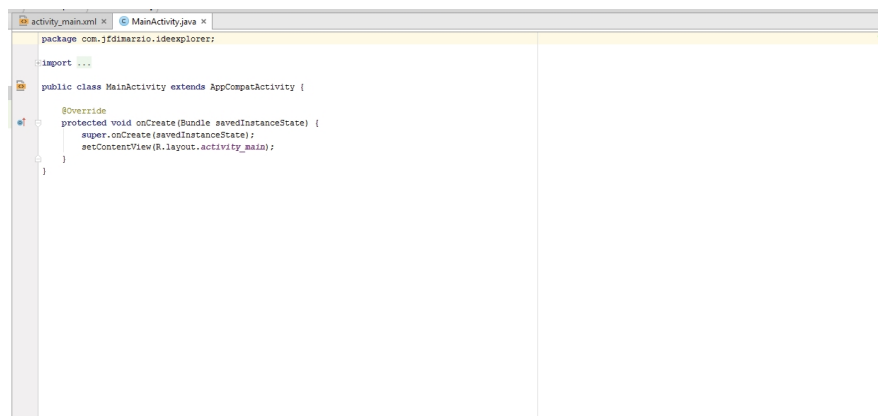
<item
name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>
</resources>

```

- **build.gradle (Module: app):** This defines the module-specific build configurations. Here you can add dependencies what you need in your Android application.

EDITOR TAB

- On the right side of the IDE (and taking up the largest area) are the Editor tabs
- The Editor tabs are where you write and work with your code files.
- To work on a new file, simply locate the file in the Project window and double-click it to open a new Editor tab that contains that file's code. If you need to create a new file from scratch, right-click the directory into which you want to place your file, and select New ⇨ *<File Type>* from the context menu.



ANDROID MONITOR

- Finally, at the bottom of the IDE, you should see a button labelled Android Monitor. Click this button to open the Android Monitor.
- The Android Monitor automatically displays when you debug an application. It contains a very useful tool called logcat. Logcat displays most of the helpful messages that are output by your application while you are trying to debug it.



Obtaining the Required Tools

Before you write your first app, you need to download the required tools.

For Android development, you can use a Mac, a Windows PC, or a Linux machine. You can freely download all the necessary tools.

Required tools for Windows 10

1. JDK
2. Android Studio
3. SDK
4. AVD

The Android Studio 2 makes use of the Java SE Development Kit 8 (JDK). If your computer does not have the JDK 8 installed, you should start by downloading it from

www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
and installing it prior to moving to the next section.

Android Studio

The first and most important piece of software you need to download is Android Studio. After you have downloaded and installed Android Studio 2, you can use the SDK Manager to download and install multiple versions of the Android SDK. Having multiple versions of the SDK available enables you to write programs that target different devices. For example, you can write one version of an application that specifically targets Android Nougat, but because that flavor of Android is on less than 1% of devices, with multiple versions of the SDK you can also write a version of your app that uses older features and targets Marshmallow or Lollipop users.

You can use the Android Device Manager to set up device emulators. You can download Android Studio 2 from <http://developer.android.com/sdk/index.html>

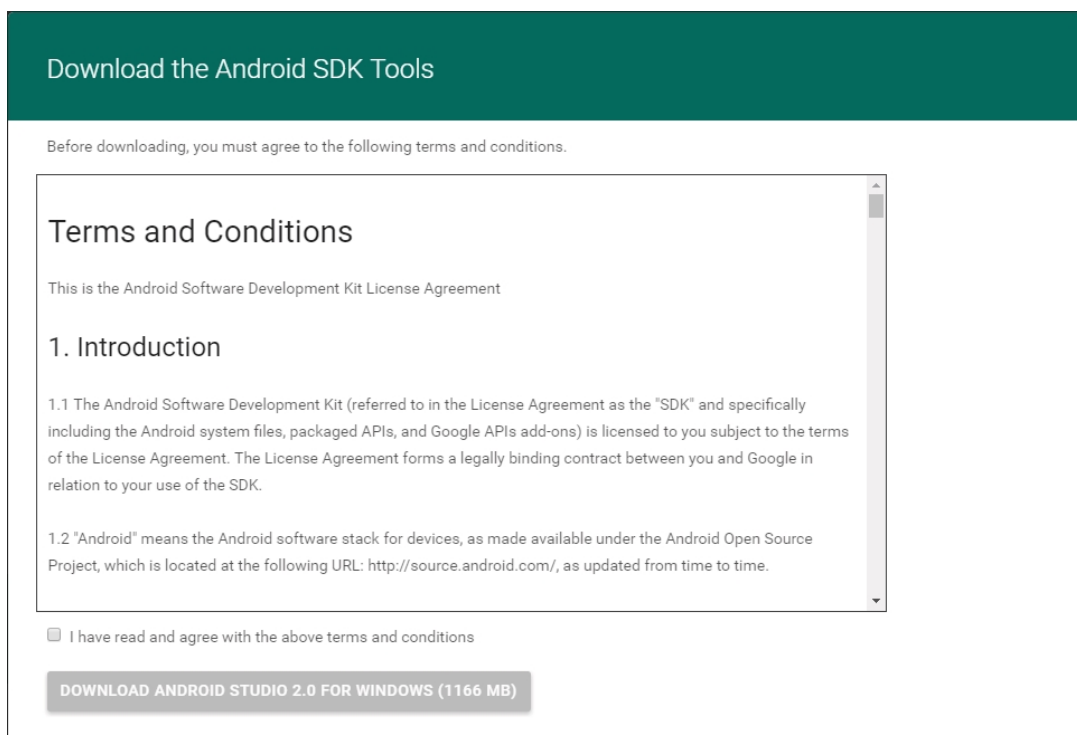
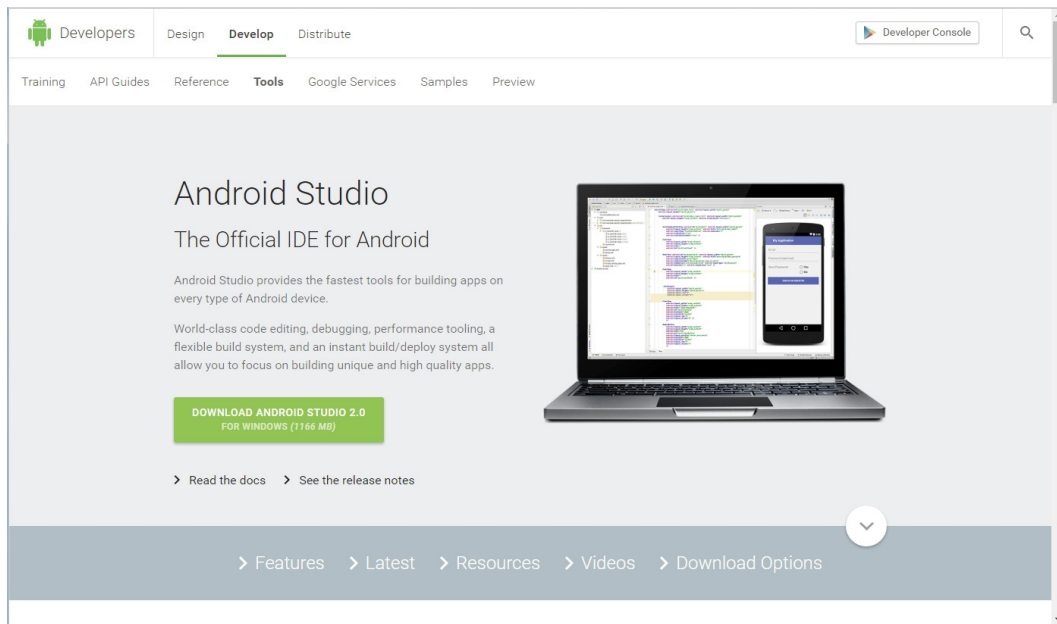
Android Studio 2 is packaged in an executable. Run the install process to set up Android Studio 2.

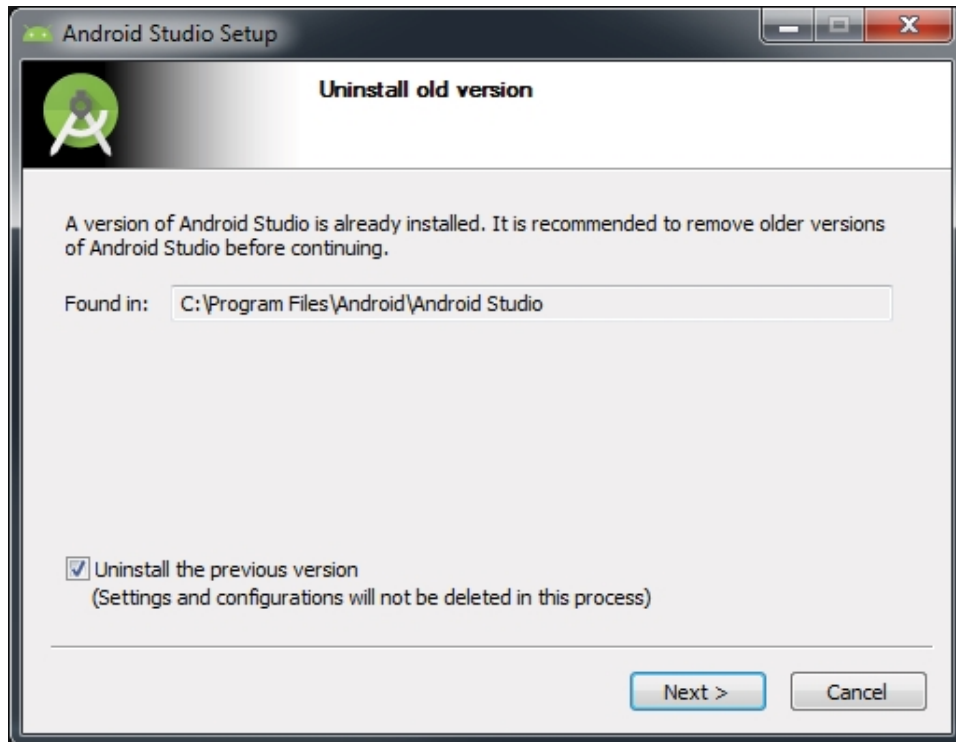
After you've downloaded and run the setup executable, use the following steps to go through the

Installation process:

1. Accept the terms and conditions shown in Figure 1-6.
2. If you have an older version of Android Studio already installed on your computer, the Android Studio Setup prompts you to automatically uninstall it. Even though the old version of Android Studio will be uninstalled, the settings

and configurations are retained. You have an opportunity to reapply those settings and configurations to Android Studio after the setup has completed.

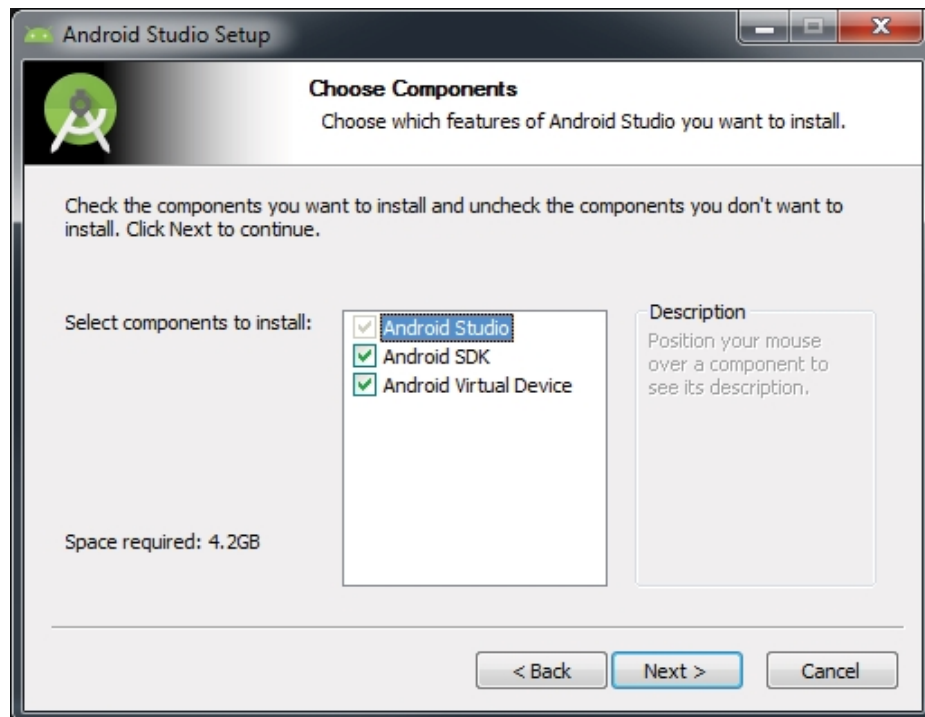




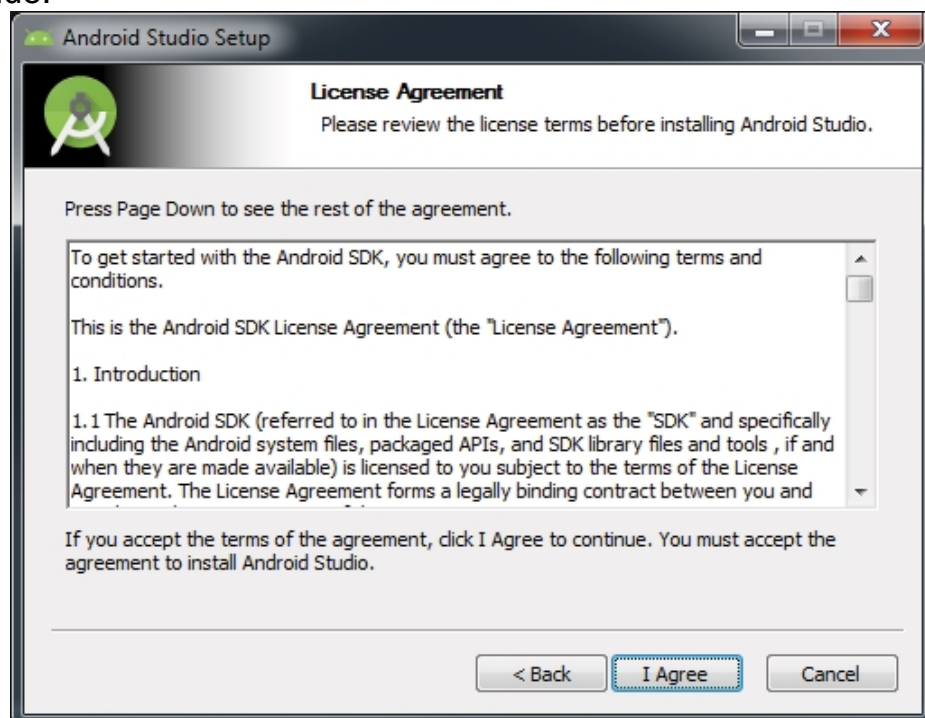
3. Click Next on the Welcome to Android Studio Setup screen



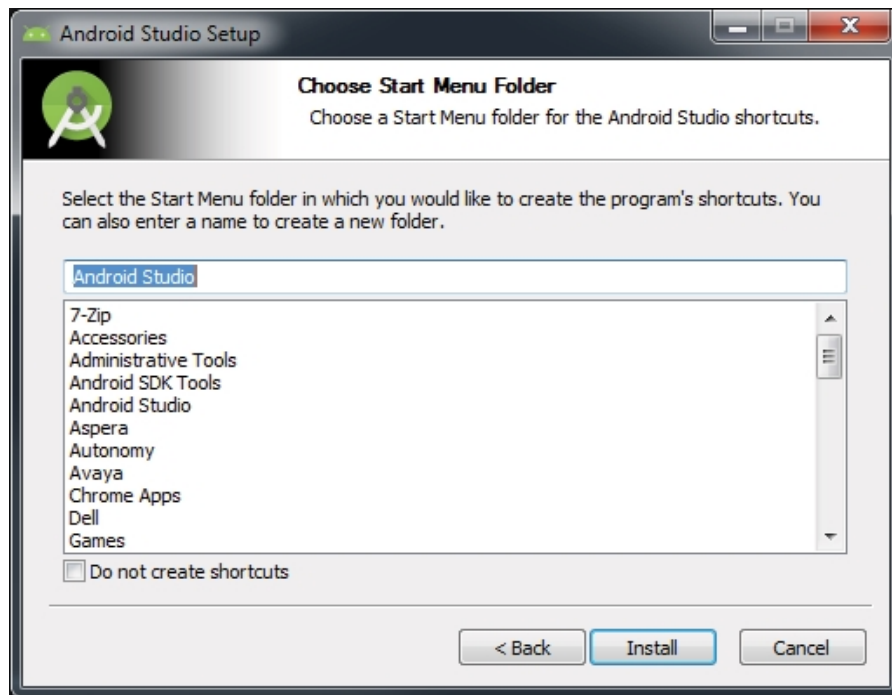
4. Pick which components of Android Studio you want to install. Android Studio is selected by default (and cannot be deselected, Android SDK and Android Virtual Device are also selected by default. Click Next to accept the default choices and continue.



5. You are presented with the License Agreement. Click I Agree to continue.



6. On the configuration settings screen, it is best to accept the default locations specified by the setup process and click Next to continue. You see the Choose Start Menu Folder screen. Click Install to kick off the Android Studio 2 installation.



7. Installing Android Studio 2 could take a few minutes, depending on the speed of your computer.

You are presented with a progress bar to help you track the state of the installation.

Android Studio 2 is installed with a default SDK (Software Development Kit), in this case Marshmallow.

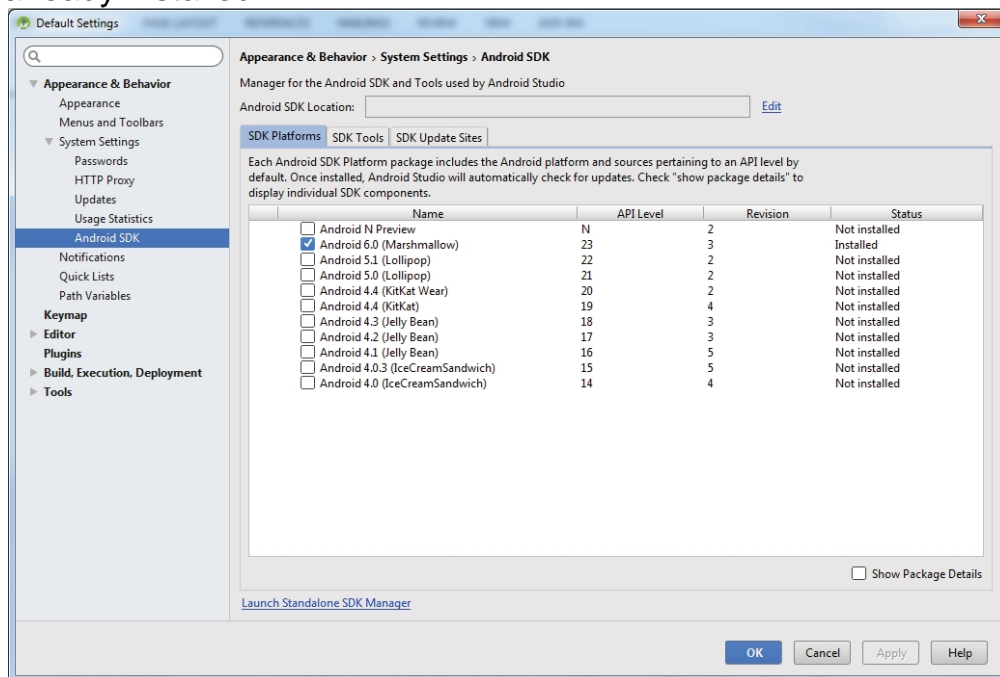
8. When the install is complete, you will see a Completing Android Studio Setup.

Leave the Start Android Studio box checked and click Finish.



Android SDK

- The most important piece of software you need to download is, of course, the Android SDK.
- The Android SDK contains all of the packages and tools required to develop a functional Android application.
- The SDKs are named after the version of Android OS to which they correspond.
- By default, the Marshmallow SDK was installed with Android Studio 2.
- However, if you want to install a different Android SDK, you can do so using the SDK Manager.
- from the Android Studio welcome screen . From this screen, click the Configure drop-down menu in the lower-right corner.
- The Configure selection menu opens. Choose SDK Manager from this menu.
- The SDK configuration screen, shows that the Marshmallow SDK is already installed.



- The setup process for Android Studio is now complete. The next section explains how to set up an
- Android Virtual Device that you can use to test your applications.

Creating Android Virtual Devices (AVDs)

- The next step is to create an Android Virtual Device (AVD) you can use for testing your Android applications.
- An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile; a mapping to a system image; and emulated storage, such as a secure digital (SD) card.

- One important thing to remember about emulators is that they are not perfect.
- There are some applications, such as games (which are GPU heavy) or applications that use sensors such as the GPS or accelerometer. These types of applications cannot be simulated with the same speed or consistency within an emulator as they can when running on an actual device.
- However, the emulator is good for doing some generalized testing of your applications.
- You can create as many AVDs as you want to test your applications with different configurations.

Use the following steps to create an AVD. This example demonstrates creating an AVD (put simply, an Android emulator) that emulates an Android device running Android N on the Nexus 5x hardware specs.

1. Start Android Studio so that the Welcome screen is visible. Click Start a New Android Studio Project. Set up a HelloWorld project. Type **Chapter1HelloWorld** in the Application Name field.
2. You can keep the default values for the other fields on the New Project . Click Next.
3. You should see the Targeted Android Devices screen. By default, the Create New Project
4. Wizard selects for you the Android SDK level that has the greatest activity based on statistics gathered from Google Play. For now, accept the default, and click Next.
5. On the Add an Activity to Mobile screen, accept the default choice—Empty Activity and click Next.

Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK: API 18: Android 4.3 (Jelly Bean)

Lower API levels target more devices, but have fewer features available.
By targeting API 18 and later, your app will run on approximately 74.3% of the devices that are active on the Google Play Store.
[Help me choose](#)

☐ Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ TV

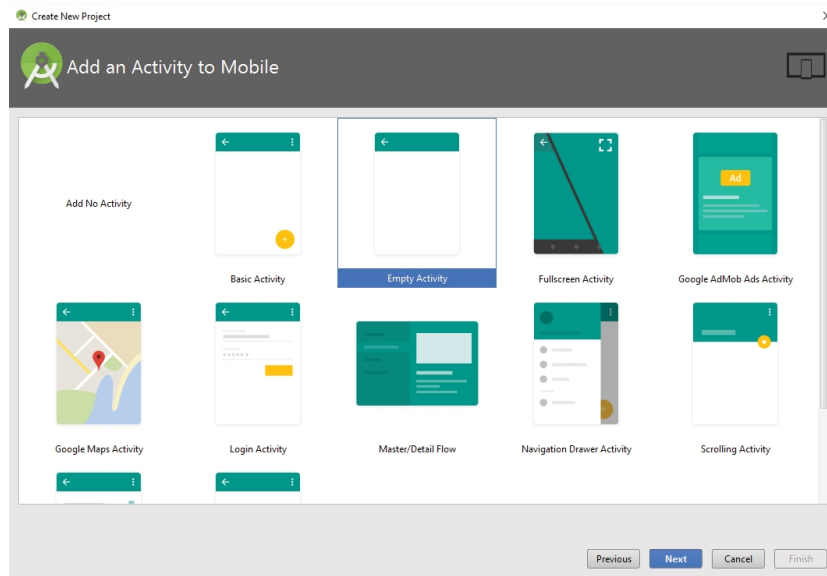
Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Android Auto

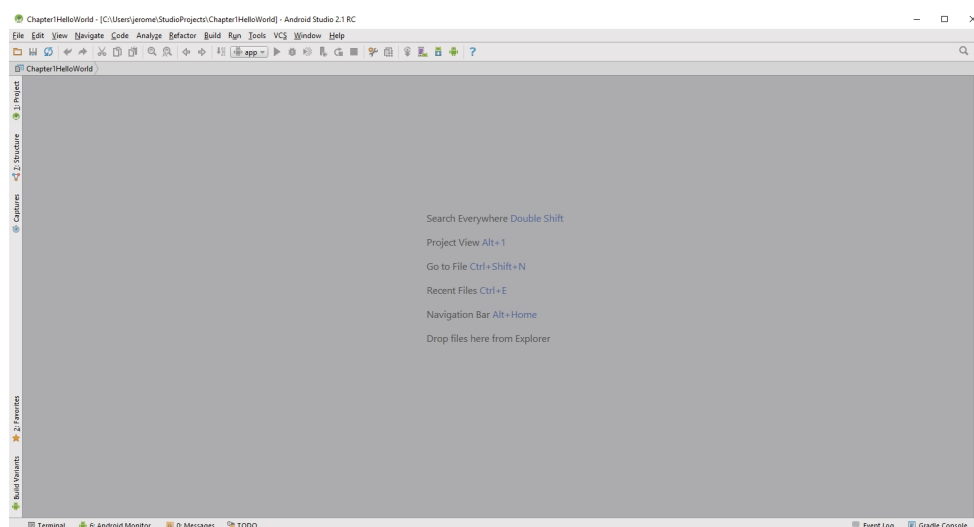
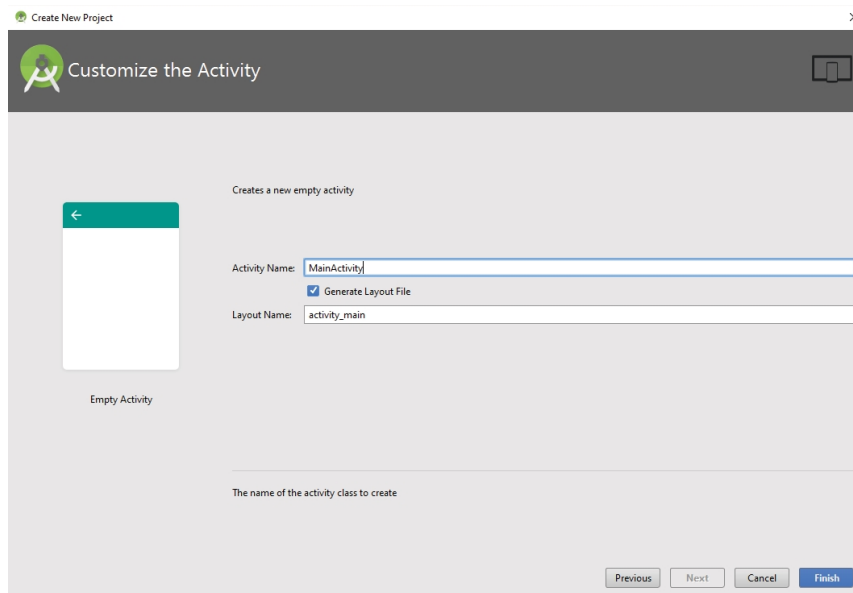
☐ Glass

Minimum SDK: Glass Development Kit Preview (API 19)

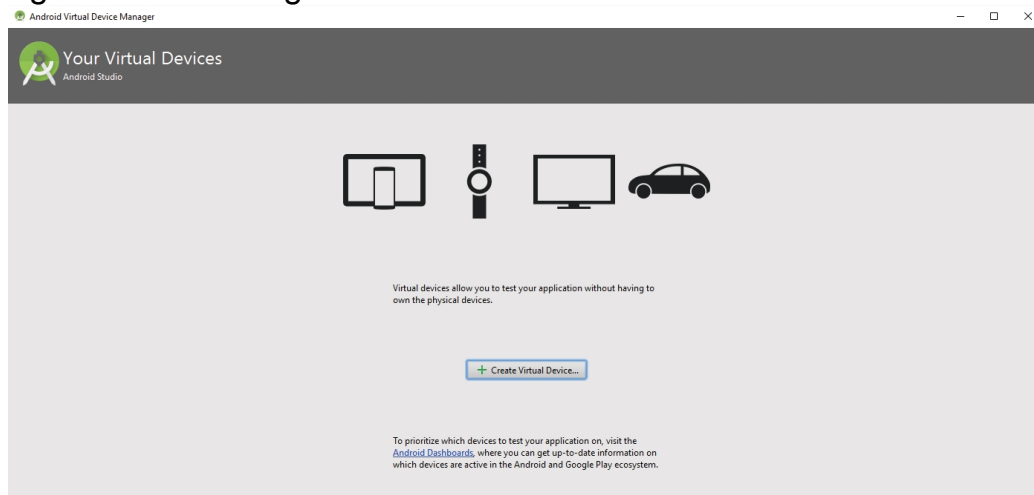
Previous Next Cancel Finish



6. Accept all of the defaults on the Customize the Activity screen, and click Finish.



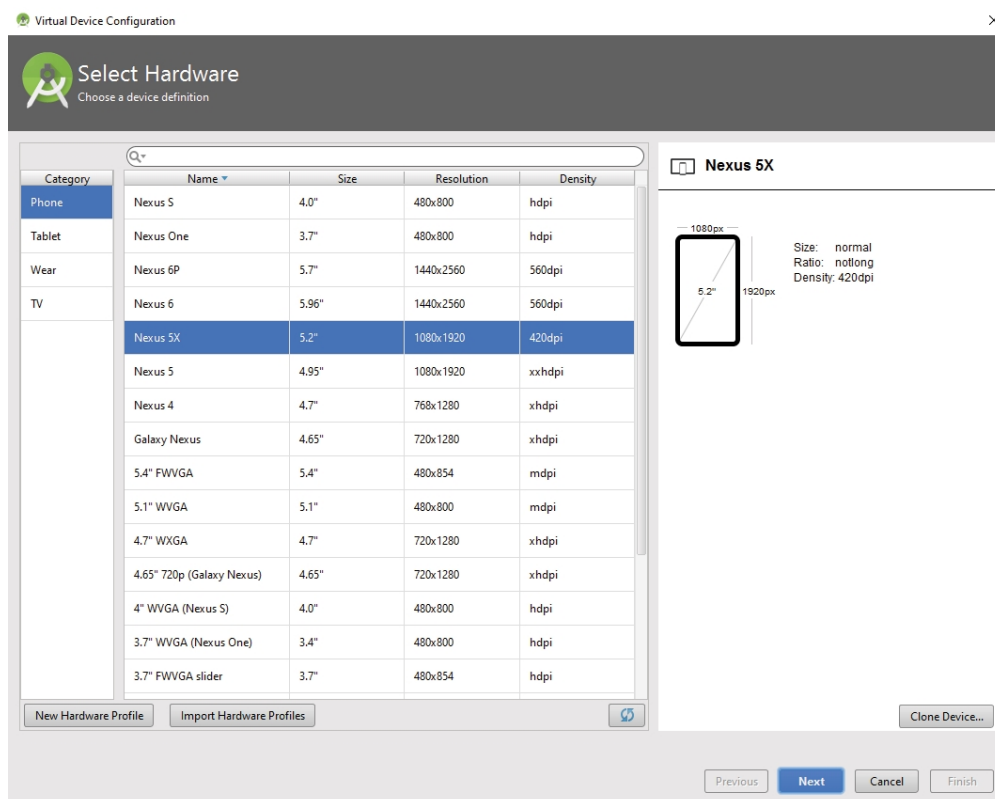
7. Launch the AVD Manager by selecting Tools ⇨ Android ⇨ AVD Manager or using the AVD Manager button from the toolbar.



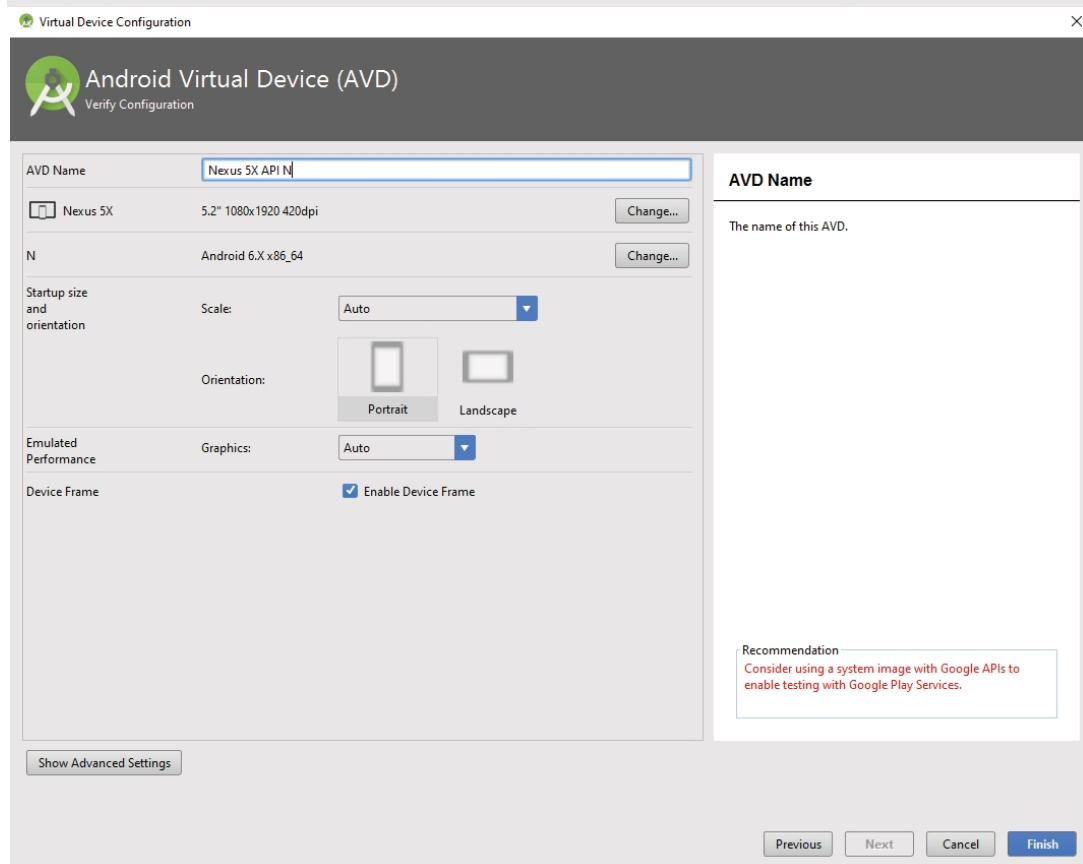
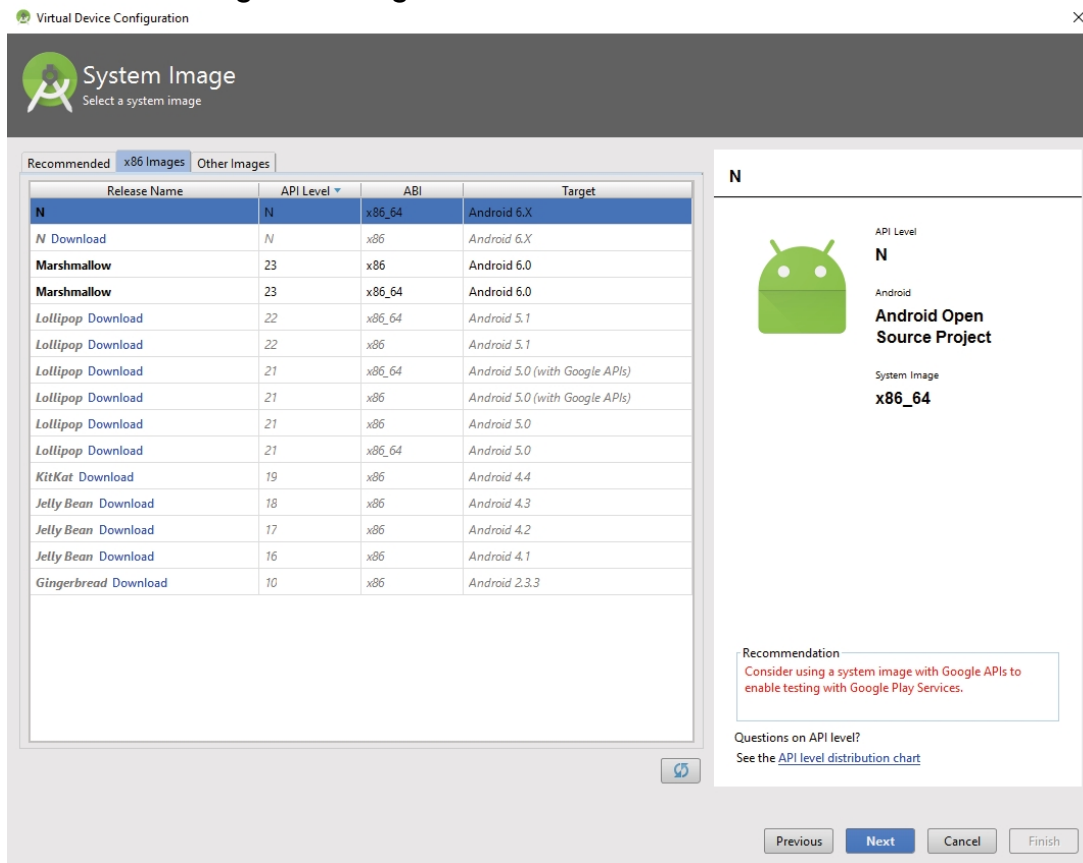
8. Click the + Create Virtual Device button to create a new AVD. The Virtual Device Configuration screen opens.

9. Select the Nexus 5x hardware profile and click Next. Although none of the emulators offers the same performance as its actual hardware counterpart, the Nexus 5x should run well on most x86-based desktops, and it still offers some of the mid- to high-end Android device specs.

10. For the system image, select and install the latest option, which at the time this book was written is Android Nougat. Click the x86 Images select N from the



11. In the Android Virtual Device (AVD) dialog, accept the defaults. Click the Finish button to begin building the AVD.



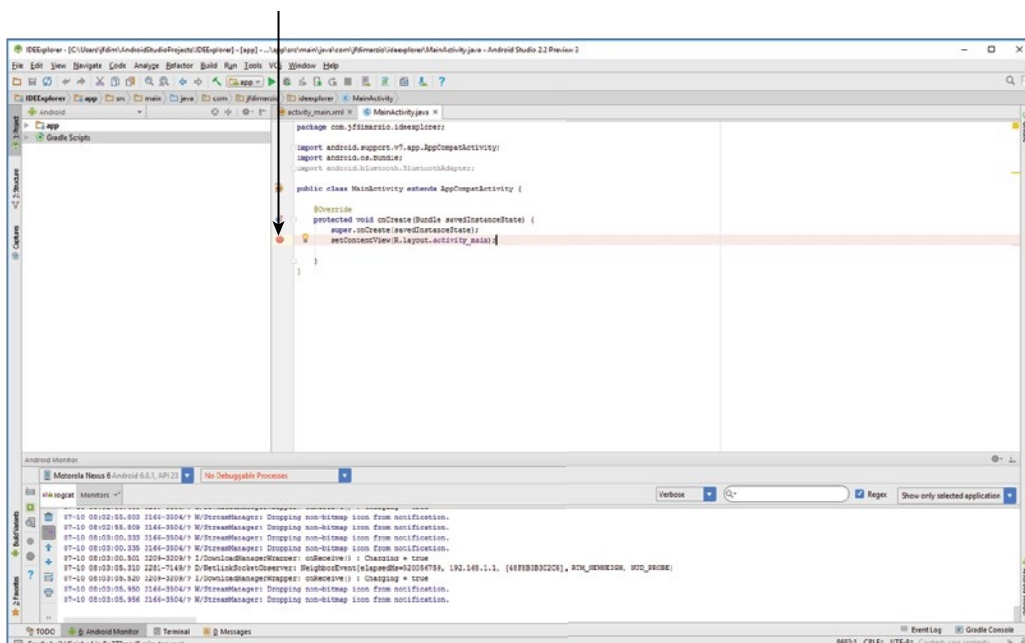
Debugging Your Application

Debugging is the process of finding and fixing errors or bugs in the source code of any software.

- After you have built an application, you need to be able to debug it and see what is going on inside your code.
- One of the handiest ways to be able to see inside your code it through the use of breakpoints.
- Breakpoints allow you to pause the execution of your code at specific locations and see what is going on (or what is going wrong).

Setting Breakpoints

- Breakpoints are a mechanism by which you can tell Android Studio to temporarily pause execution of your code, which allows you to examine the condition of your application.
- This means that you can check on the values of variables in your application while you are debugging it. Also, you can check whether certain lines of code are being executed as expected—or at all.
- To tell Android Studio that you want to examine a specific line of code during debugging, you must set a breakpoint at that line.
- Click the margin of the editor tab next to line of code you want to break at, to set a breakpoint. A red circle is placed in the margin, and the corresponding line is highlighted in red.



- You can also set a breakpoint by placing your cursor in the line of code where you want it to break and clicking Run ⇌ Toggle Line Breakpoint.

Notice that the term used is *toggle*, which means that any breakpoints you set can be turned off the same way you turn them on.

- Simply click an existing breakpoint to remove it from your code.
- *Android Studio only pauses execution at breakpoints when you debug your application—not when you run it.*

METHOD Breakpoint

- You can set a method breakpoint by selecting Run ⇌ Toggle Method Breakpoint. A method breakpoint is represented by a red circle containing four dots placed at the method signature.

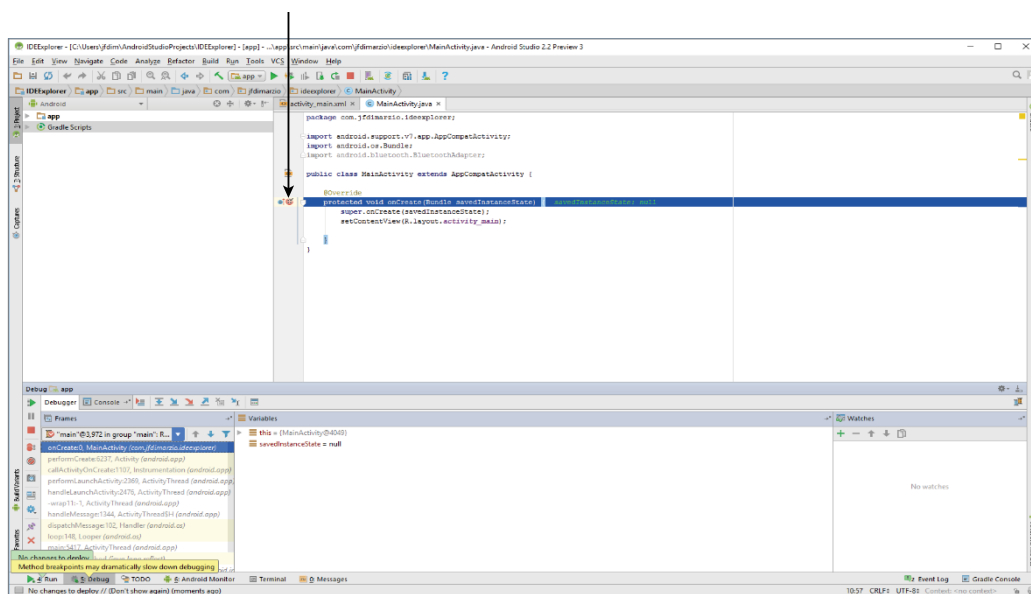


Figure 2-12

Temporary Breakpoints

- A temporary breakpoint is useful when you are trying to debug a large loop, or you just want to make sure a line of code is being hit during execution.
- To set a temporary breakpoint, place your cursor at the location in the code where you want it to break and select Run ⇌ Toggle Temporary Line Breakpoint.
- Notice that a red circle containing a 1 is now placed in the margin
- The 1 in the red circle represents the fact that Android Studio only stops at this breakpoint the first time your code enters it.
- After that, the line is executed as though there is no breakpoint set. This can be very useful if you want to ensure a line within a loop is being hit, but you don't want to stop at the line every time it is executed.

Conditional Breakpoints

- A condition breakpoint is a breakpoint at which Android Studio only pauses when specific conditions are met.
- To set a conditional breakpoint, first set a simple breakpoint at the line of code you want to examine, then right-click the simple breakpoint to bring up the condition context menu.
- From here you can set conditions that tell Android Studio when to pause at a breakpoint.
- For example, you can tell Android Studio to only pause at a line of code when your variable named foo equals true. You would then set the condition in the breakpoint to

`foo == true`

- Conditional breakpoints are extremely useful in diagnosing intermittent issues in complex code blocks.

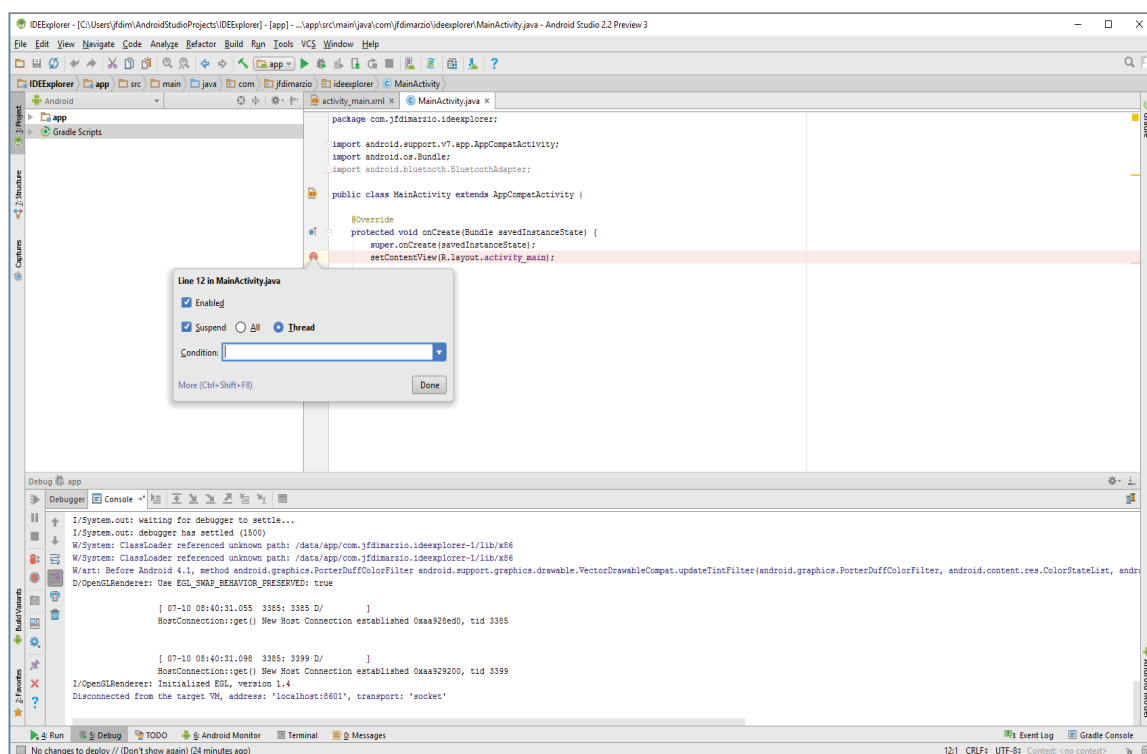
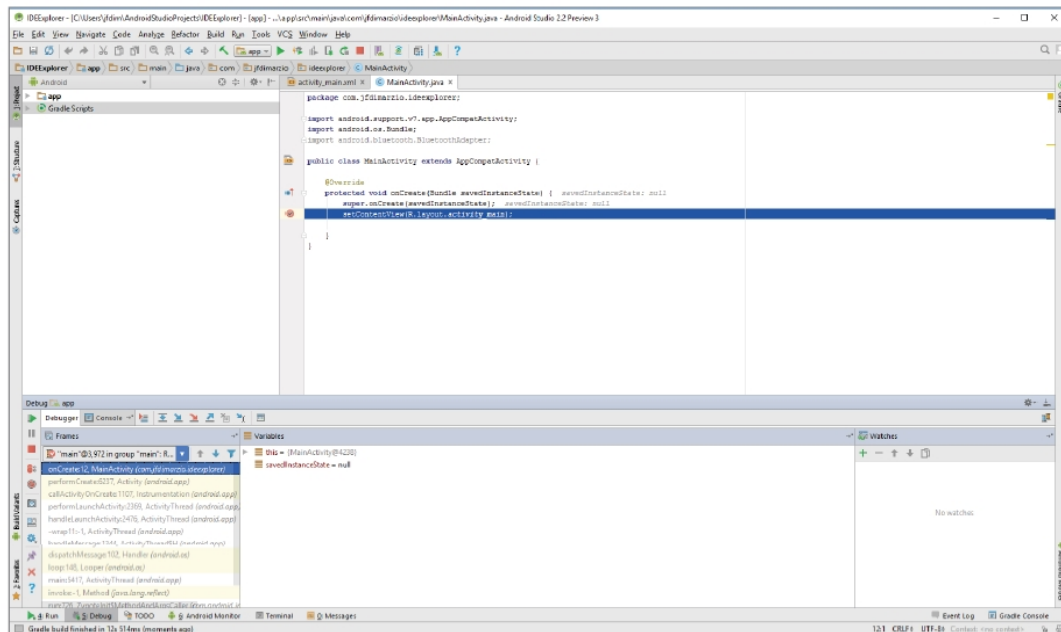


Figure 2-14

Navigating Paused Code

- While in debug mode, Android Studio pauses at any breakpoint that you have set. That is, as long as a breakpoint has been set on a reachable line of code (a line of code that would be executed by system), Android Studio halts execution at that line until you tell it to continue.

- When Android Studio hits, and pauses at, a breakpoint, the red circle in the margin next to the corresponding line of code changes to a circle with a check mark.



- Once a breakpoint has been hit, the debug window opens at the bottom of Android Studio.
- The debug window contains many of the tools you use to navigate around your code.



- Notice the navigation buttons located in the menu bar of the debug window. The most commonly used are Step Over and Step Into.
- Step Over advances you to the line of code that immediately follows the one at which you are currently paused.
- This means that if you are paused at a method call, and you press Step Over, Android Studio executes the method call without pausing and then pauses again when execution reached the next line.
- Step Into follows execution wherever it leads in the code(i.e breakpoint that we set) if you are paused at a method call and click Step Into, Android Studio will shift the view to the method call and pause execution at the first line of code within that method. This allows you to then follow the execution of that method line-by-line before it returns to the calling block.

Publishing Your Application

After you have created, and fully debugged, your application, you might want to deploy it to the Google Store for others to enjoy. The following sections outline the steps for publishing your applications.

Generating a Signed APK

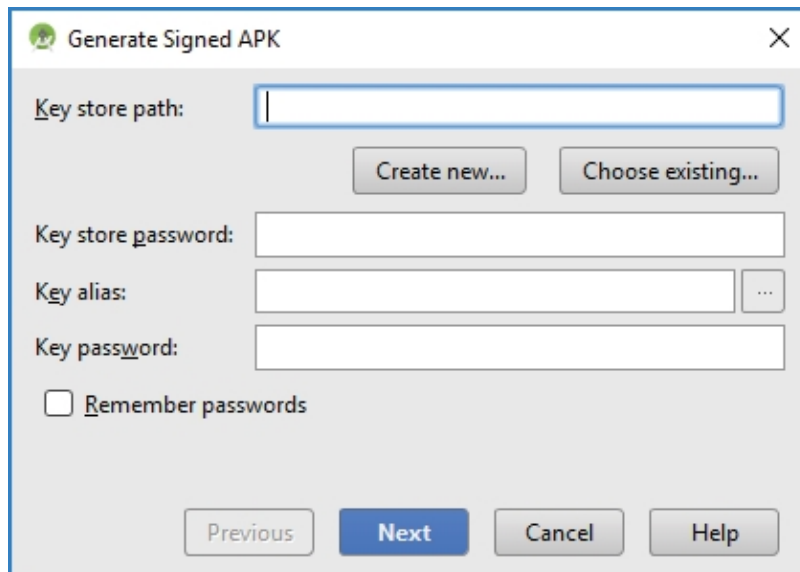
To publish your finished application on the Google Play Store, you must generate a signed APK(the Android application package).

The APK is the compiled, executable version of your application.

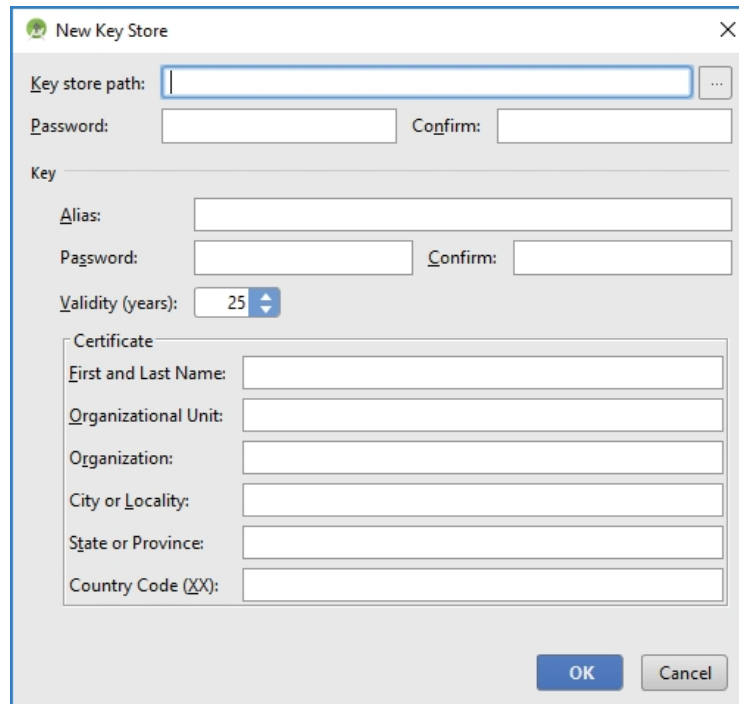
Signing it is much like signing your name to a document. The signature identifies the app's developer to Google and the users who install your application.

More importantly, unless your Android Studio is in developer mode, unsigned applications will not run. Use the following steps to generate a signed APK:

1. Generate a signed APK from your code by selecting Build ⇔ Generate Signed APK from the Menu bar to bring up the Generate Signed APK window



2. Assuming you have never published an application from Android Studio, you need to create a new key store. Click the Create New button to display the New Key Store window.
3. Fill out all of the information on this form because it pertains to your entity and application. Notice that there are two places for a password. These are the passwords for your key store and your key, respectively. Because a key store can hold multiple keys, it requires a separate password than that of the key for a specific app.



4. Click OK to return to the Generate Signed APK window.
5. In the Generate Signed APK windows, click Next to review and finish the process. Now that you have a signed APK, you can upload it to the Google Play Store using the developer console at <https://play.google.com/apps/publish/>.

Introduction to Activities in Android

- Activity class is one of the very important parts of the Android Component.
- Any app, don't matter how small it is (in terms of code and scalability), has at least one Activity class.
- Unlike most programming languages, in which the main() method is the entry point for that program or application to start its execution, the android operating system initiates the code in an Activity instance by invoking specific callback methods that correspond to specific stages of its Lifecycle.
- An activity is the entry point for interacting with the user.
- Every activity contains the layout, which has a user interface to interact with the user.
- Layout for a particular activity is set with the help of setContentView().
- setContentView() is a function/method that takes View as a parameter.

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
```

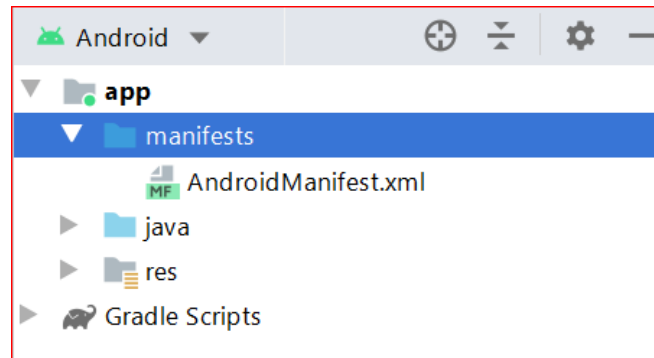
```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Declaring Activity in Manifest File

Open the app folder, and then open the subfolder manifest, and then open the **AndroidManifest.xml** file.



Let's suppose the reader wants to have one more activity, apart from the MainActivity which is included by default in the project. Before adding one more activity and not doing any changes,

Let's see what the AndroidManifest.xml File looks like

- XML

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.gfg.exampleactivity">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ExampleActivity"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

</manifest>

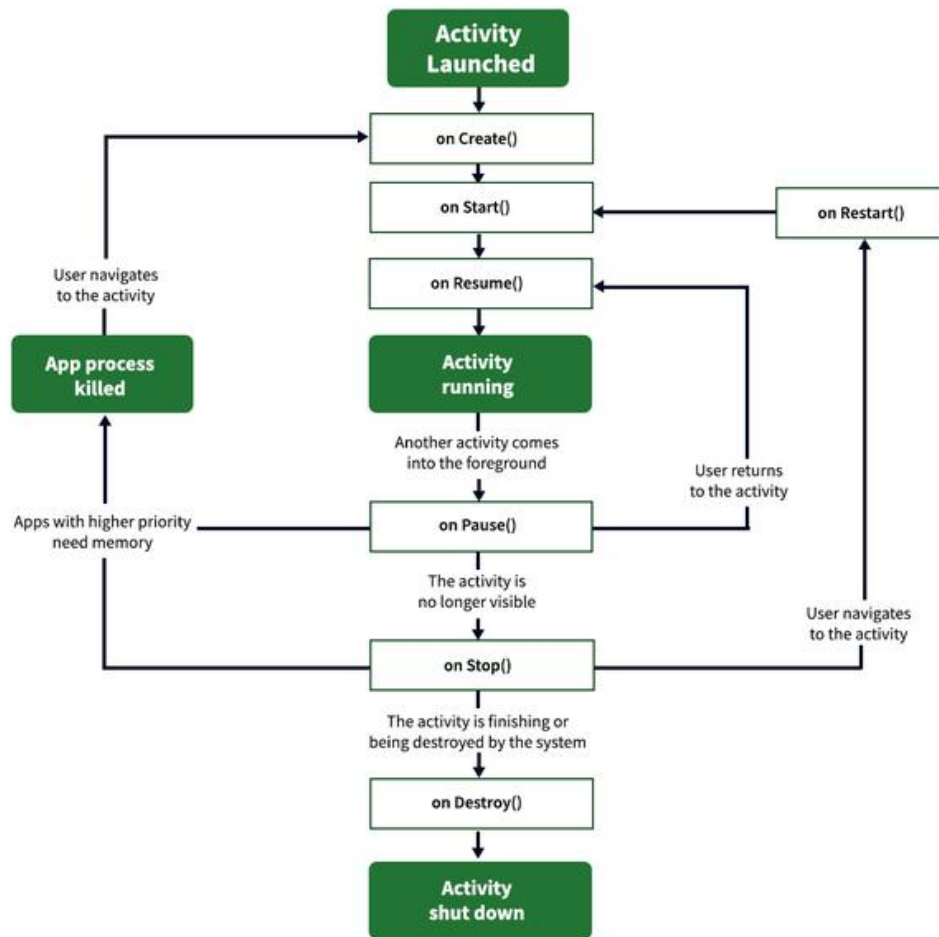
Now let's add another activity named `SecondActivity`, and see how to declare it in the manifest file. **One must write the Declaration Code within the `application` tag, otherwise, the declaration will give the error, and `SecondActivity` will not be detected by the System.** The Declaration Code is given below.

```
<activity
    android:name=". SecondActi vi ty"
    android:exported="true" >
</activity>
```

- So it can conclude that an application can have one or more activities without any restrictions.
- Every activity that the android app uses must be declared in the **AndroidManifest.xml** file.
- The main activity for the app must be declared in the manifest with a **<intent-filter>** that includes the MAIN action and LAUNCHER.
- Any activity whether it is **MainActivity** or any other activity must be declared within the **<application>** of the **AndroidManifest** file.
- If the user forgets to declare any of the activity, then android will not be able to detect that activity in the app.
- If either the MAIN action or LAUNCHER category is not declared for the main activity, then the app icon will not appear in the Home screen's list of apps.

Activity Lifecycle in Android

- 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.



Activity Lifecycle in Android

1. **onCreate()** - It is called when the activity is first created.
2. **onStart()** - It is invoked when the activity is visible to the user.
3. **onResume()** - It is invoked when the activity starts interacting with the user.
4. **onPause()** -
 - It is invoked when the activity starts interacting with the user.
 - When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen).
5. **onStop()** - It is invoked when the activity is not visible to the user.
6. **onDestroy()** - called before the activity is destroyed by the system.
7. **onRestart()** - called when the activity has been stopped and is restarting again.

Demo Android App to Demonstrate Activity Lifecycle in Android

```

package com.jfdimarzio.activity101;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends AppCompatActivity
{

```



```

String tag = "Lifecycle Step";
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(tag, "In the onCreate() event");
}
public void onStart()
{
    super.onStart();
    Log.d(tag, "In the onStart() event");
}
public void onRestart()
{
    super.onRestart();
    Log.d(tag, "In the onRestart() event");
}
public void onResume()
{
    super.onResume();
    Log.d(tag, "In the onResume() event");
}
public void onPause()
{
    super.onPause();
    Log.d(tag, "In the onPause() event");
}
public void onStop()
{
    super.onStop();
    Log.d(tag, "In the onStop() event");
}
public void onDestroy()
{
    super.onDestroy();
    Log.d(tag, "In the onDestroy() event");
}
}

```

Applying Styles and Themes to an Activity

- By default, an activity is themed to the default Android theme.
- However, there has been a push in recent years to adopt a new theme known as Material.
- The Material theme has a much more modern and cleaner look to it.
- There are two versions of the Material theme available to Android developers:

- Material Light and
 - Material Dark.
- Either of these themes can be applied from the **AndroidManifest.xml**.
- To apply one of the Material themes to an activity, simply modify the **<Application>** element in the AndroidManifest.xml file by changing the default android:theme attribute. (Please be sure to change all instances of "com.jfdimarzio" to whatever package name your project is using.)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.jfdimarzio.activity101">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@android:style/Theme.Material">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Changing the default theme to @android:style/Theme.Material, as in the highlighted code in the preceding snippet, applies the Material Dark theme and gives your application a darker look.

Hiding the Activity Title

You can also hide the title of an activity if desired (such as when you just want to display a status update to the user).

To do so, use the requestWindowFeature() method and pass it the Window.FEATURE_NO_TITLE constant, like this:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Window;

public class MainActivity extends AppCompatActivity
{
    @Override
```

```

        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            requestWindowFeature(Window.FEATURE_NO_TITLE);
        }
    }

```

Now you need to change the theme in the **AndroidManifest.xml** to a theme that has no title bar. Be sure to change all instances of "com.jfdimarzio" to whatever package name your project is using.

```

package com.jfdimarzio.activity101;
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.jfdimarzio.activity101">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.NoTitleBar">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```



This hides the title bar



Android - Intents and Filters

An Android **Intent** is an abstract description of an operation/Action to be performed.

The **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action.

Intent object is used to call other activities.

Intent object:

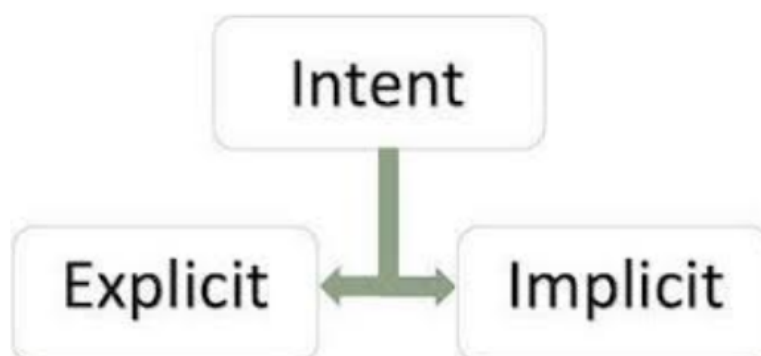
```
startActivity(new Intent("com.jfdimarzio.SecondActivity"));
```

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

TYPE OF INTENT

Intents are of two types:



[Intent](#) are the objects which is used in android for passing the information among Activities in an Application and from one app to another also.

[Intent](#) are used for communicating between the Application components and it also provides the connectivity between two apps.

Methods and their Description

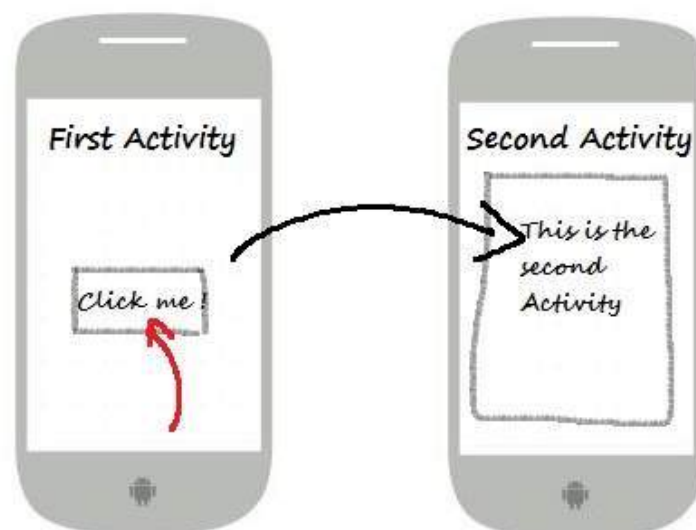
Methods	Description
<code>Context.startActivity()</code>	This is to launch a new activity or get an existing activity to be action.
<code>Context.startService()</code>	This is to start a new service or deliver instructions for an existing service.
<code>Context.sendBroadcast()</code>	This is to deliver the message to broadcast receivers.

EXPLICIT INTENT

Explicit intents are communicated between two activities inside the same application. We can use explicit intents when we need to move from one activity to another activity.

Example:-

1. when a user wants to start a service to download a file or when a new activity gets started in response to a user action.
2. In amazon app if you go to Home page you can see prime, fresh, mobies, electronics etc. If you click prime it transit to prime page activity like wise other tab also works.



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

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

- In Explicit we use the name of component which will be affected by Intent. For
- Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.
- Explicit Intent work internally within an application to perform navigation and data transfer.
- The below given code snippet will help you understand the concept of Explicit Intents

```
// Explicit Intent by specifying its class name
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
// Starts TargetActivity
startActivity(intent);
```

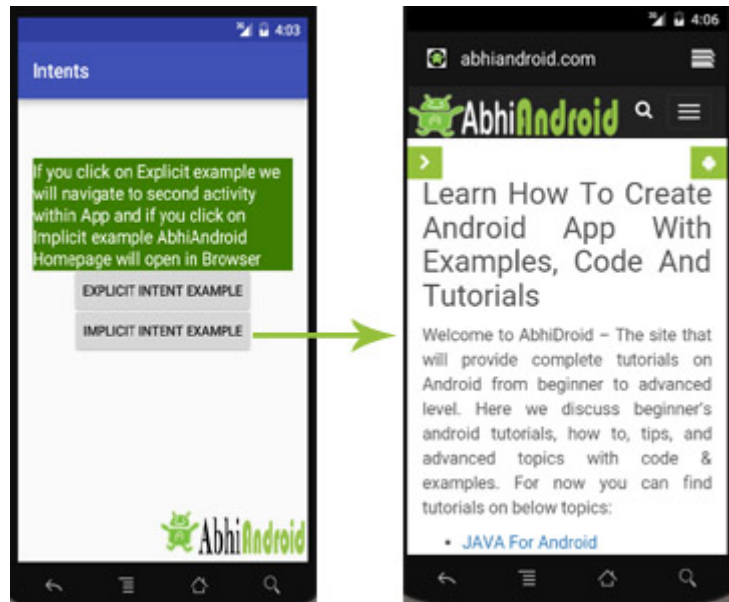
IMPLICIT INTENT

- In Implicit Intents we do not need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.
- The basic example of implicit Intent is to open any web page
- Let's take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

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

(OR)

```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("https://www.abhiandroid.com"));
startActivity(intentObj);
```



For some intents, there is no need to specify the data. For example, to select a contact from the Contacts application, you specify the action and then indicate the MIME type using the setType() method:

```
Intent i = new Intent(android.content.Intent.ACTION_PICK);  
i.setType(ContactsContract.Contacts.CONTENT_TYPE);
```

The setType() method explicitly specifies the MIME data type to indicate the type of data to return.

(A media type (also known as a Multipurpose Internet Mail Extensions or MIME type))

Intent Filters

<intent-filter> element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

```
<activity android:name=".CustomActivity"  
  android:label="@string/app_name">  
  
  <intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <action android:name="com.example.My Application.LAUNCH" />  
    <category android:name="android.intent.category.DEFAULT" />  
    <data android:scheme="http" />  
  </intent-filter>  
  
</activity>
```


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

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

There are following test Android checks before invoking an activity –

A filter **<intent-filter>** may list more than one action as shown above but this list cannot be empty; a filter must contain at least one **<action>** element, otherwise it will block all intents.

A filter **<intent-filter>** may list zero, one or more than one categories.

Each **<data>** element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme**, **host**, **port**, and **path** for each part of the URI.

Android Fragments

Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity. In [Android](#), the fragment is the part of [Activity](#) which represents a portion of User Interface(UI) on the screen.

Following are important points about fragment –

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- Fragments can't live on their own. They must be *hosted* by an activity or another fragment.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

The **FragmentManager** class is responsible to make interaction between fragment objects.

The Java class for a fragment needs to extend the Fragment base class:

```
public class Fragment1 extends Fragment {  
    }
```

Adding Fragments dynamically

To add fragments to an activity, you use the FragmentManager class by first obtaining an instance of it:

```
FragmentManager fragmentManager = getSupportFragmentManager();
```

You also need to use the FragmentTransaction class to perform fragment transactions (such as add, remove, or replace) in your activity:

```
FragmentTransaction fragmentTransaction =  
fragmentManager.beginTransaction();
```

In this example, the **WindowManager** is used to determine whether the device is currently in portrait mode or landscape mode.

Once that is determined, you can add the appropriate fragment to the activity by creating the fragment.

Next, you call the **replace()** method of the FragmentTransaction object to add the fragment to the specified view container.

In this case, android.R.id.content refers to the content view of the activity.

```
//---landscape mode---
```

```
Fragment1 fragment1 = new Fragment1();
```

```
// android.R.id.content refers to the content view of the activity
```

```
fragmentTransaction.replace(  
    android.R.id.content, fragment1);
```

Using the replace() method is essentially the same as calling the remove() method followed by the add() method of the FragmentTransaction object.

To ensure that the changes take effect, you need to call the commit() method:

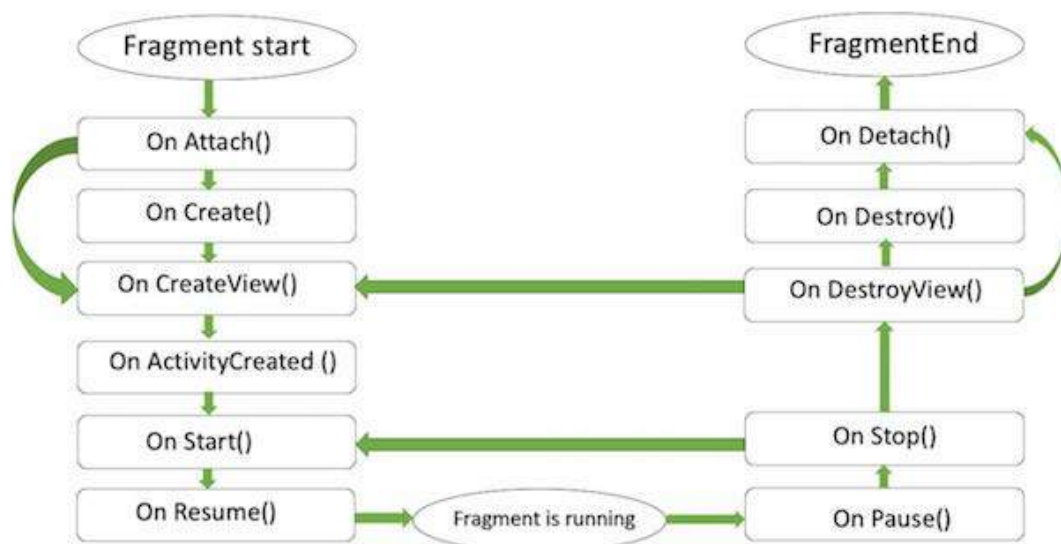
```
fragmentTransaction.commit();
```

Types of Android Fragments

1. **Single Fragment:** Display only one single view on the device screen. This type of fragment is mostly used for mobile phones.
2. **List Fragment:** This Fragment is used to display a list-view from which the user can select the desired sub-activity. The menu drawer of apps like Gmail is the best example of this kind of fragment.
3. **Fragment Transaction:** This kind of fragments supports the transition from one fragment to another at run time. Users can switch between multiple fragments like switching tabs.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity.



Android Fragment Lifecycle Methods

No.	Method	Description
1)	onAttach(Activity)	it is called only once when it is attached with activity.
2)	onCreate(Bundle)	It is used to initialize the fragment.
3)	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.

4)	onActivityCreated(Bundle)	It is invoked after the completion of onCreate() method.
5)	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart()	makes the fragment visible.
7)	onResume()	makes the fragment interactive.
8)	onPause()	is called when fragment is no longer interactive.
9)	onStop()	is called when fragment is no longer visible.
10)	onDestroyView()	allows the fragment to clean up resources.
11)	onDestroy()	allows the fragment to do final clean up of fragment state.
12)	onDetach()	It is called immediately prior to the fragment no longer being associated with its activity.