

INTRODUCTION

Contents

- Background
- Android Versions
- Features
- Architecture
- Obtaining the required tools
- Launching Android Applications
- Exploring Android Studio IDE
- Android code completion
- Debugging application
- Publishing application

Background

- Android is a mobile operating system that is based on a modified version of Linux.
- Android OS is Open and Free.
- The main advantage to adopting Android is that it offers a unified approach to application development.
- Developers need only develop for Android in general, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android.

Android Versions



Android Versions (Contd...)

Android Version	Release date	Codename
1.0	September 23, 2008	Alpha
1.1	February 9, 2009	Beta
1.5	April 30, 2009	Cupcake
1.6	September 15, 2009	Donut
2.0/2.1	October 26, 2009	Éclair
2.2	May 20, 2010	Froyo
2.3	December 6, 2010	Gingerbread
3.0/3.1/3.2	February 22, 2011	Honeycomb
4.0	October 18, 2011	Ice Cream Sandwich

Android Versions (Contd...)

Android Version	Release date	Codename
4.1	July 9, 2012	Jelly Bean
4.4	October 31, 2013	KitKat
5.0	November 12, 2014	Lollipop
6.0	October 5, 2015	Marshmallow
7.0	August 22, 2016	Nougat
8.0	August 21, 2017	Oreo
9.0	August 6, 2018	Pie

Android Versions (Contd...)

Code name	Version number	Linux kernel version ^[3]	Initial release date	API level
(No codename) ^[4]	1.0	?	September 23, 2008	1
Petit Four ^[4]	1.1	2.6	February 9, 2009	2
Cupcake	1.5	2.6.27	April 27, 2009	3
Donut ^[5]	1.6	2.6.29	September 15, 2009	4
Eclair ^[6]	2.0 – 2.1	2.6.29	October 26, 2009	5 – 7
Froyo ^[7]	2.2 – 2.2.3	2.6.32	May 20, 2010	8
Gingerbread ^[8]	2.3 – 2.3.7	2.6.35	December 6, 2010	9 – 10
Honeycomb ^[9]	3.0 – 3.2.6	2.6.36	February 22, 2011	11 – 13
Ice Cream Sandwich ^[10]	4.0 – 4.0.4	3.0.1	October 18, 2011	14 – 15
Jelly Bean ^[11]	4.1 – 4.3.1	3.0.31 to 3.4.39	July 9, 2012	16 – 18
KitKat ^[12]	4.4 – 4.4.4	3.10	October 31, 2013	19 – 20
Lollipop ^[13]	5.0 – 5.1.1	3.16	November 12, 2014	21 – 22
Marshmallow ^[14]	6.0 – 6.0.1	3.18	October 5, 2015	23
Nougat ^[15]	7.0 – 7.1.2	4.4	August 22, 2016	24 – 25
Oreo ^[16]	8.0 – 8.1	4.10	August 21, 2017	26 – 27
Pie ^[17]	9.0	4.4.107, 4.9.84, and 4.14.42	August 6, 2018	28
Legend: Old version Older version, still supported Latest version				

Key changes in Android 7.0

- Split-screen multi-window mode
- Redesigned notification shade
- Refined “Doze” feature
- Switch from JRE (Java Runtime Environment) to OpenJDK

Features of Android

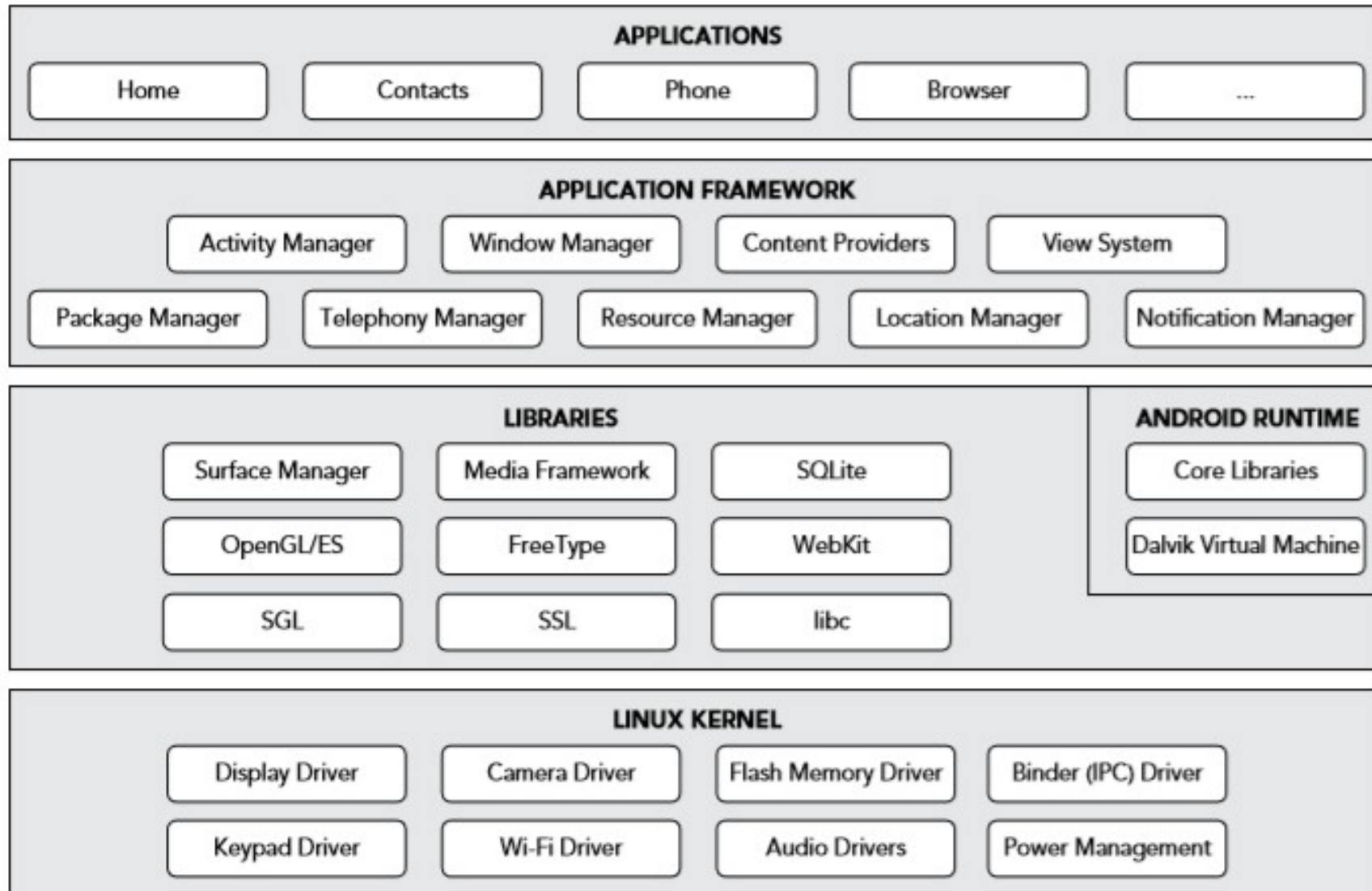
- Storage—SQLite, a lightweight relational database, for data storage.
- Connectivity—GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), Wi-Fi, LTE, and WiMAX.
- Messaging—Both SMS and MMS.
- Media support H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.

Features of Android(Contd..)

- Hardware support—Accelerometer sensor, camera, digital compass, proximity sensor, and GPS.
- Multi-touch—Multi-touch screens.
- Multi-tasking—Multi-tasking applications.
- Tethering—Sharing of Internet connections as a wired/wireless hotspot.

Android's web browser is based on the open source WebKit and Chrome's V8 JavaScript engine.

Architecture of Android



Architecture of Android(Contd...)

Android operating system is a stack of software components which is roughly divided into five sections and four main layers.

- Linux kernel—This is the kernel on which Android is based. This layer contains all the low-level device drivers for the various hardware components of an Android device.
- Libraries—These contain the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

Key core Android libraries available to the Android developer

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

Architecture of Android(Contd...)

- Android runtime—The Android runtime is located in the same layer with the libraries and provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. (Android applications are compiled into Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU power.

Architecture of Android(Contd...)

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

The Android framework includes the following key services –

- ✓ **Activity Manager** – Controls all aspects of the application lifecycle and activity stack.
- ✓ **Content Providers** – Allows applications to publish and share data with other applications.
- ✓ **Resource Manager** – Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- ✓ **Notifications Manager** – Allows applications to display alerts and notifications to the user.
- ✓ **View System** – An extensible set of views used to create application user interfaces.
- **Applications**—At this top layer are the applications that ship with the Android device (such as Phone, Contacts, Browser, and so on), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

Android Devices in the Market

Android devices come in all shapes and sizes including, but not limited to, the following types of devices:

- Smartphones
- Tablets
- E-reader devices
- Internet TVs
- Automobiles
- Smartwatches

The Android Market

- Users can simply use the Google Play application that is preinstalled on their Android devices to directly download third-party applications to their devices. Both paid and free applications are available in the Google Play Store, although paid applications are available only to users in certain countries because of legal issues.

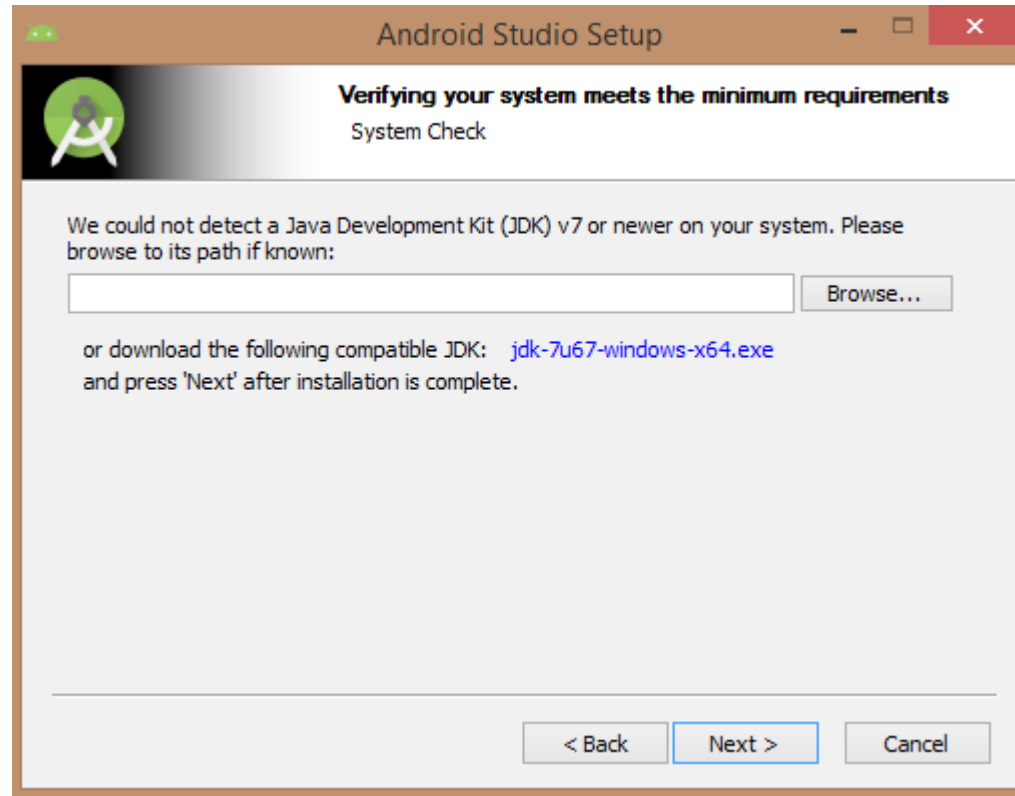
Obtaining the required tools

- Java JDK (Can be downloaded from Oracle website)
- Android Studio(developer.android website)

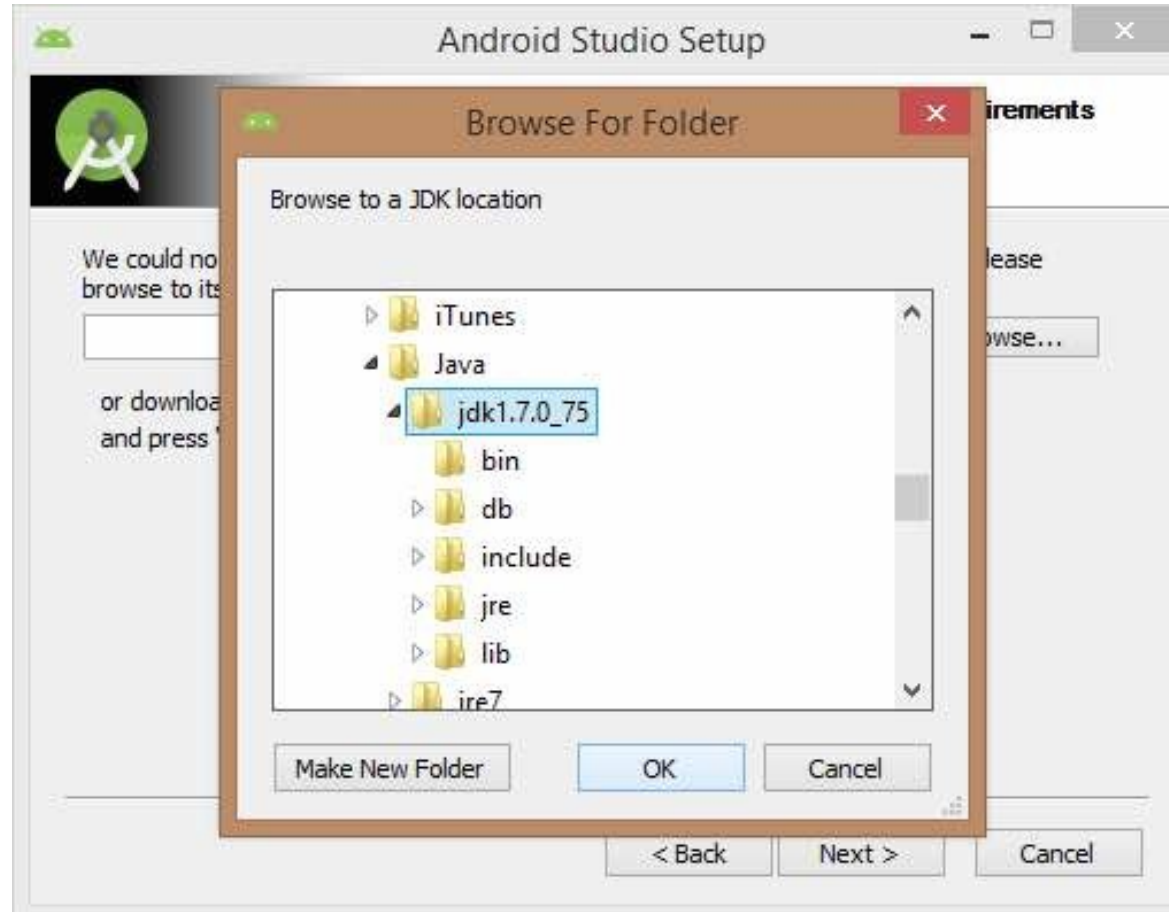
Android Studio 2.2 Installation Steps



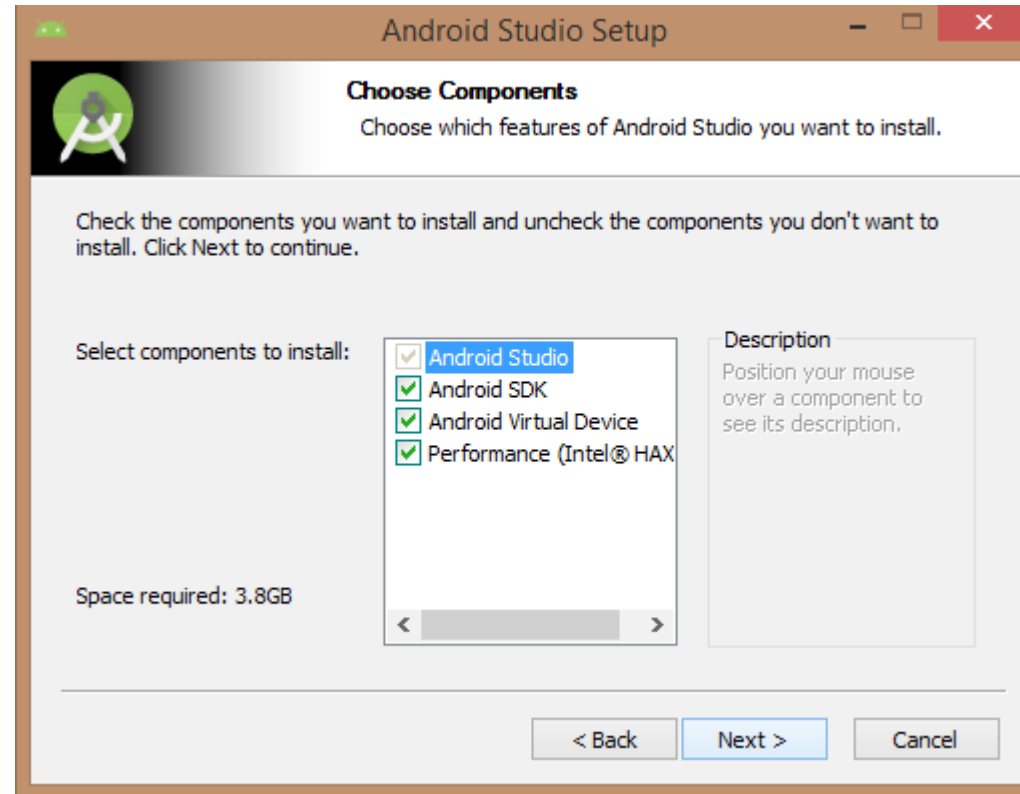
Specify JDK Path



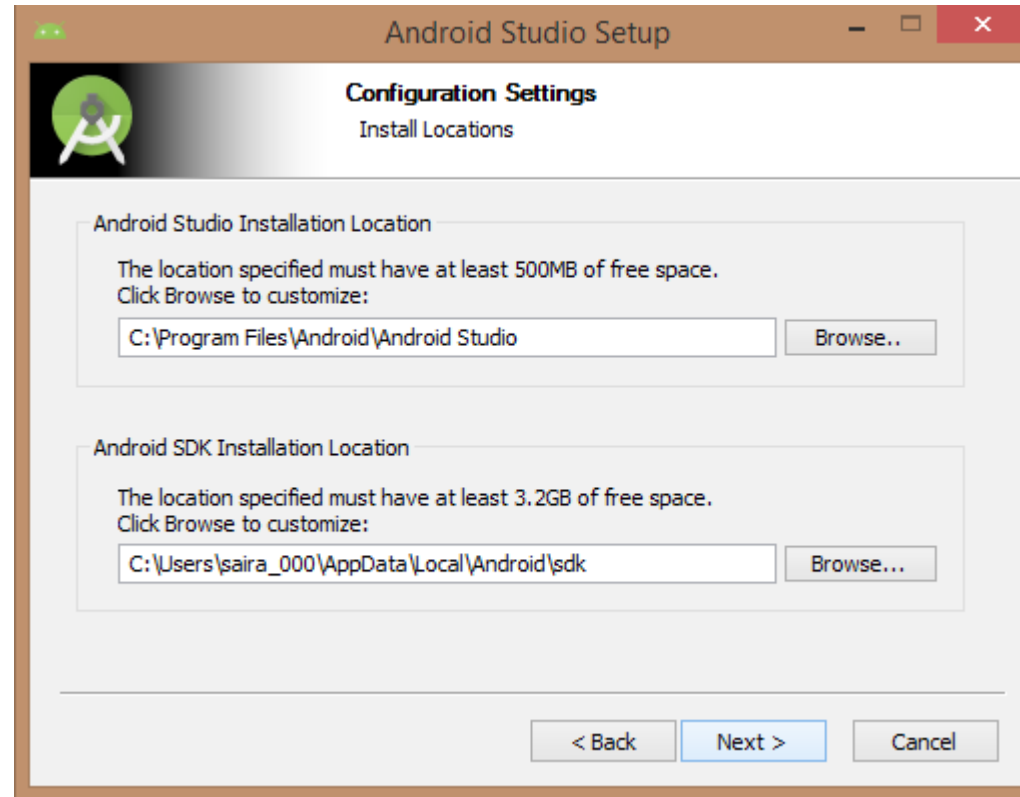
Specify JDK Path (Contd...)



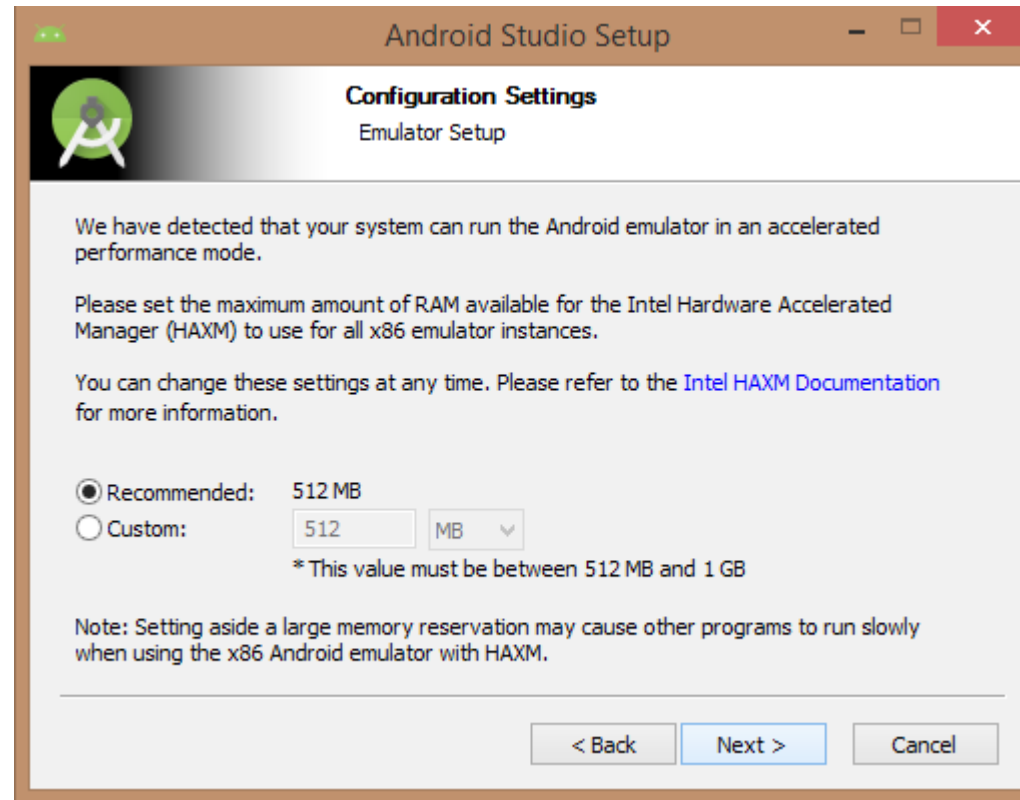
Required components selection to create applications



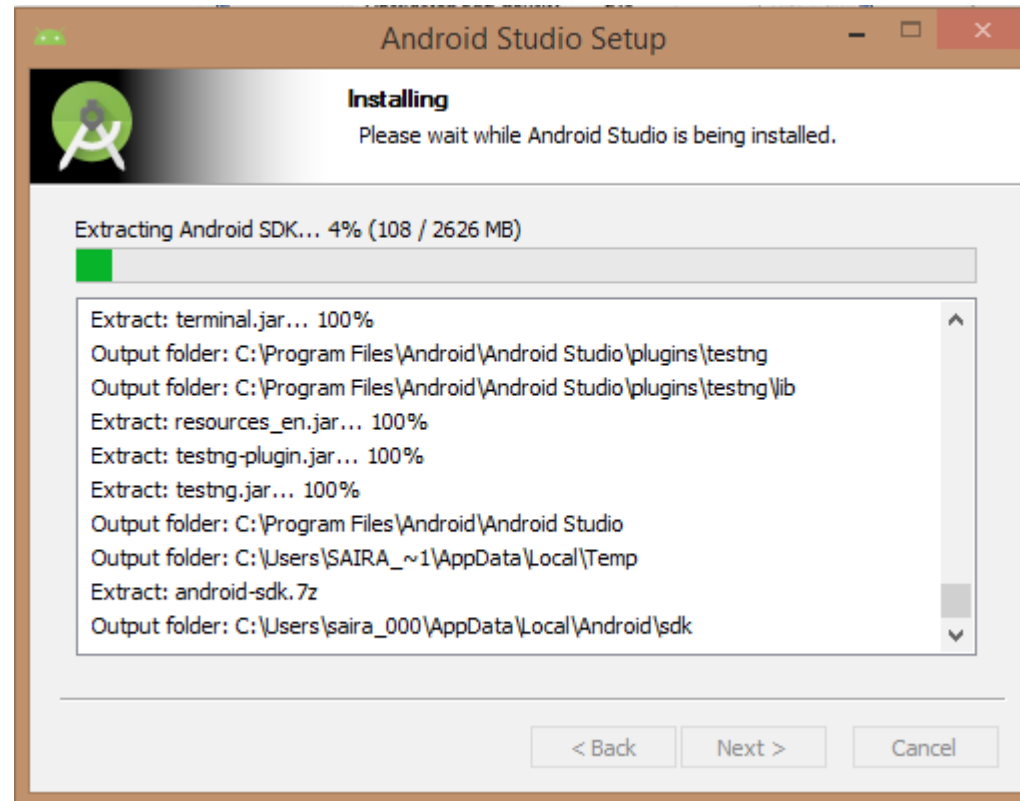
Specify the location of local machine path for Android studio and Android SDK



Specify the ram space for Android emulator by default it would take 512MB of local machine RAM

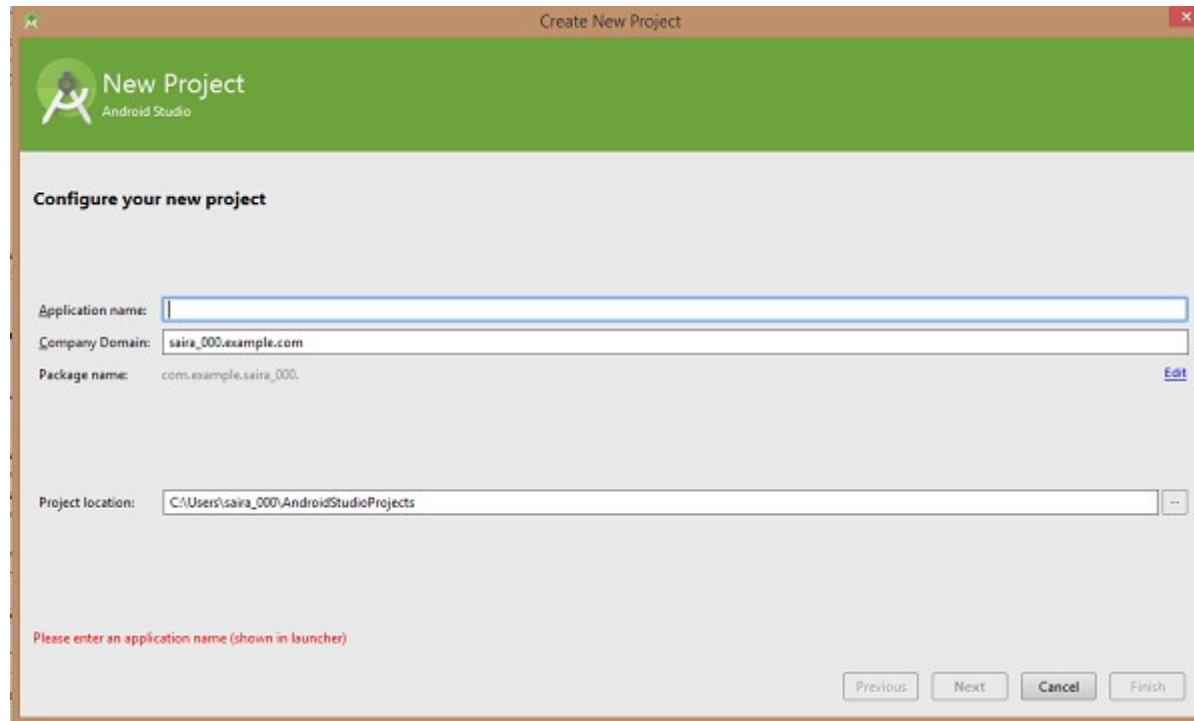


Extract SDK packages into our local machine





Creating New Android Studio Project



The screenshot shows the 'Create New Project' dialog box in Android Studio. The window has a title bar that says 'Create New Project'. Inside, there's a green header with the Android Studio logo and the text 'New Project' and 'Android Studio'. Below the header, the main area is titled 'Configure your new project'. It contains four input fields: 'Application name' (empty), 'Company Domain' (filled with 'saira_000.example.com'), 'Package name' (filled with 'com.example.saira_000.' and has an 'Edit' link to its right), and 'Project location' (filled with 'C:\Users\saira_000\AndroidStudioProjects' and has a folder selection icon to its right). At the bottom left, there is a red error message: 'Please enter an application name (shown in launcher)'. At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

Create New Project

New Project
Android Studio

Configure your new project

Application name:

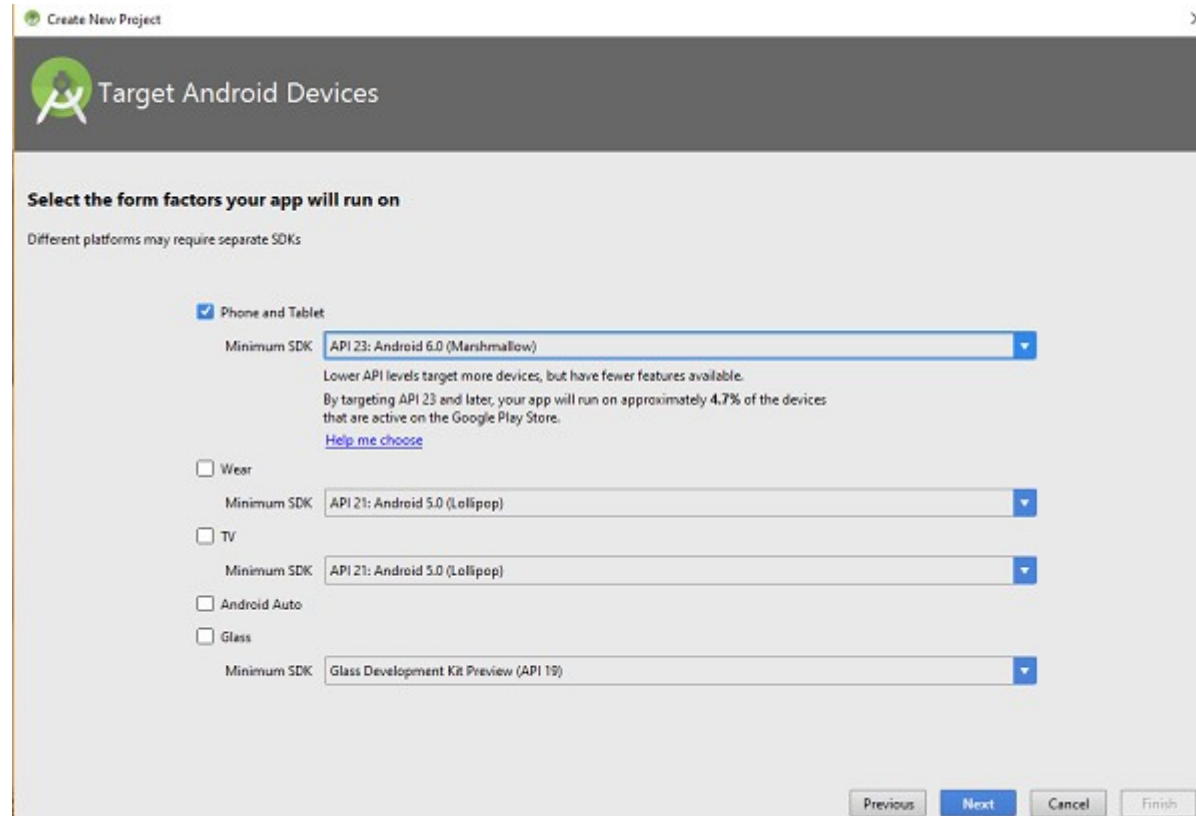
Company Domain:

Package name: [Edit](#)

Project location:

Please enter an application name (shown in launcher)

Selecting Target Android Devices



Create New Project

Target Android Devices

Select the form factors your app will run on

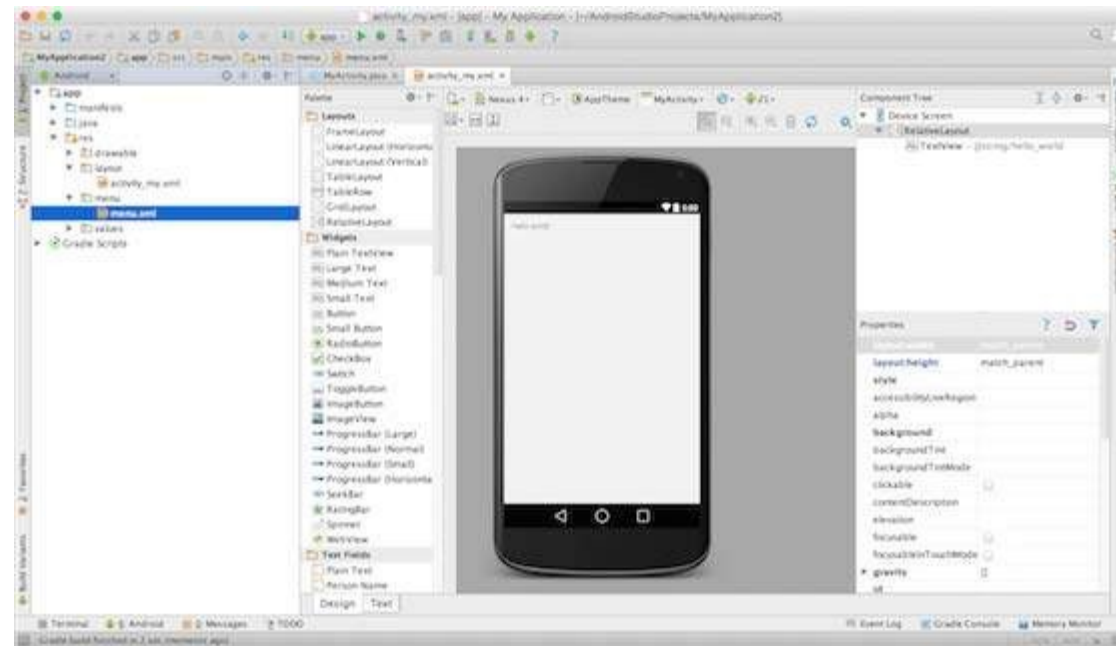
Different platforms may require separate SDKs

- ☒ Phone and Tablet
 - Minimum SDK: API 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)

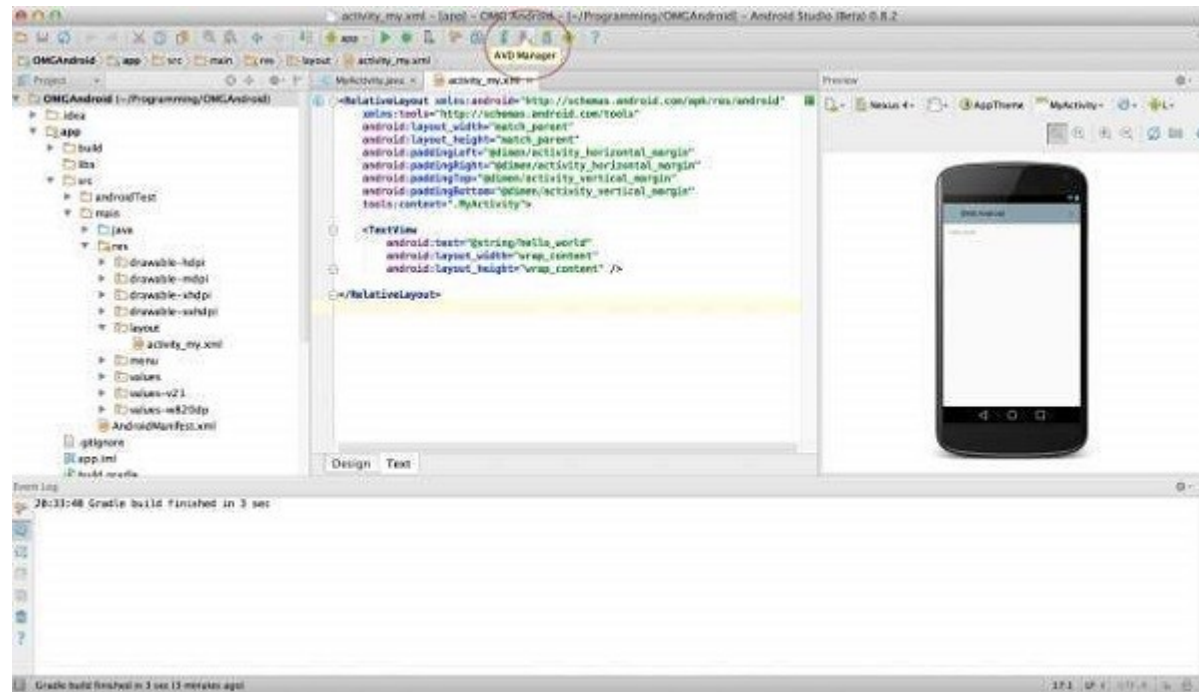
Previous Next Cancel Finish



Development Tools



Create Android Virtual Device



Creating a Jellybean emulator

1. Launch the AVD Manager by selecting Tools ➤ ⇄ ➤ Android ➤ ⇄ ➤ AVD Manager or using the AVD Manager button from the toolbar.
2. In the Android Virtual Device Manager Wizard, click the + Create Virtual Device button.
3. Select the Nexus 5x hardware profile and click Next.
4. 4. Click the x86 Images tab, select Jelly Bean from the list of images, and then click Download.
5. 5. Accept the agreement and download the Jelly Bean SDK.
6. After the SDK has downloaded, click Jelly Bean once again (on the x86 Images tab) and click Next.
7. In the Android Virtual Device (AVD) dialog, accept the defaults and click the Finish button.

Android Virtual Device Manager



Your Virtual Devices

Android Studio

VT-x is disabled in BIOS.

[Troubleshoot](#)

Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus SX API 23	1080 x 1920 420dpi	23	Android 6.0 (Google APIs)	x86	2 GB	

Create Virtual Device...



The Android Developer Community

The following are some developer communities and websites that you can turn to for help if you run into problems while working with Android:

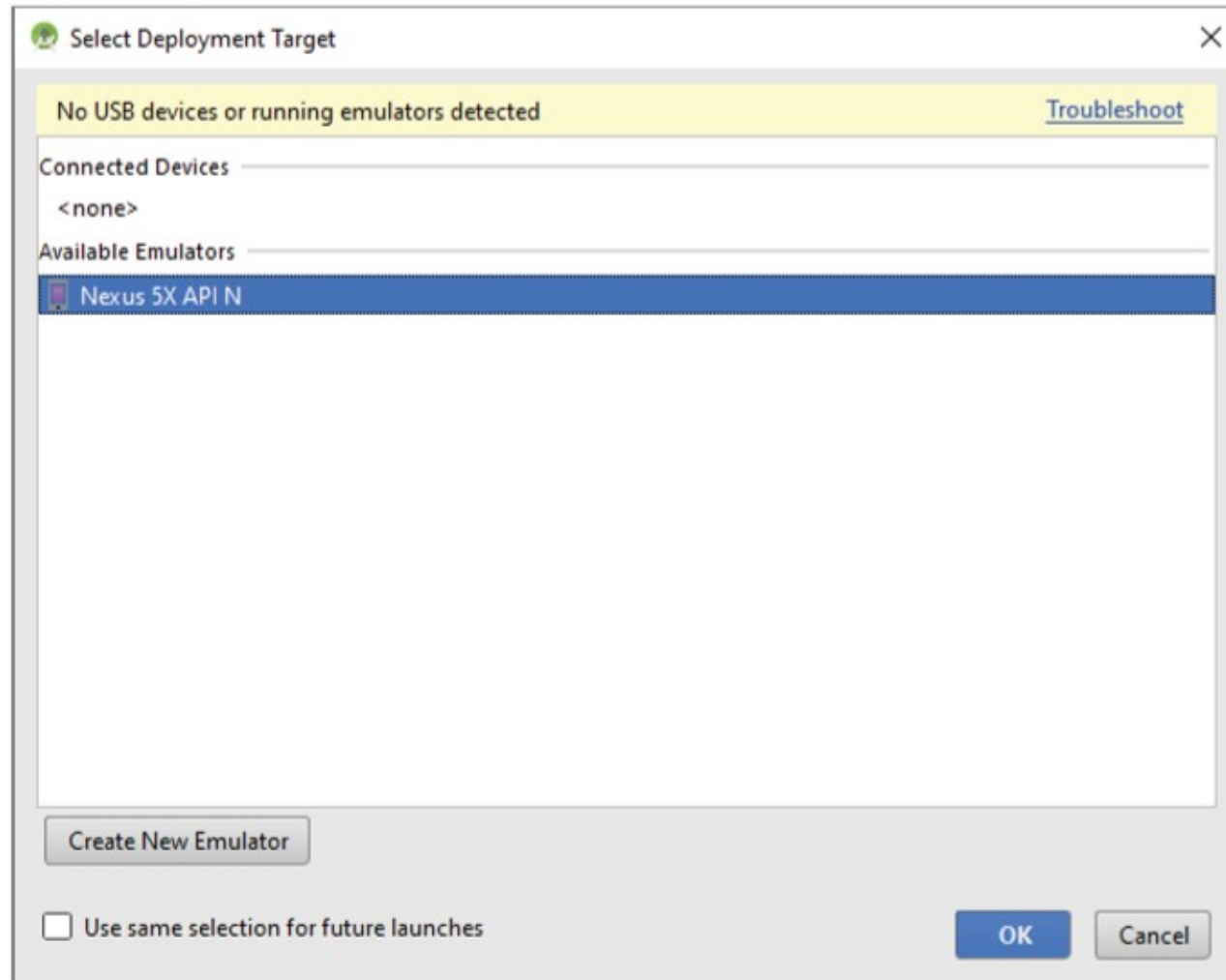
- Stack Overflow (www.stackoverflow.com)—Stack Overflow is a collaboratively edited question-and-answer site for developers. If you have a question about Android, chances are someone at Stack Overflow is probably already discussing the same question. It's also likely that someone else has already provided the answer. Best of all, other developers can vote for the best answer so that you can know which are the answers that are most trustworthy.

The Android Developer Community(Contd...)

- Google Android Training (<http://developer.android.com/training/index.html>)—Google has launched the Android Training site, which contains a number of useful classes grouped by topics. At the time of writing, the classes mostly contain code snippets that are useful to Android developers who have started with the basics.
- Android Discuss (<http://groups.google.com/group/android-discuss>)—Android Discuss is a discussion group hosted by Google using the Google Groups service. Here, you will be able to discuss the various aspects of Android programming. This group is monitored closely by the Android team at Google, so this is good place to clarify your doubts and to learn new tips and tricks.

Launching Your First Android Application

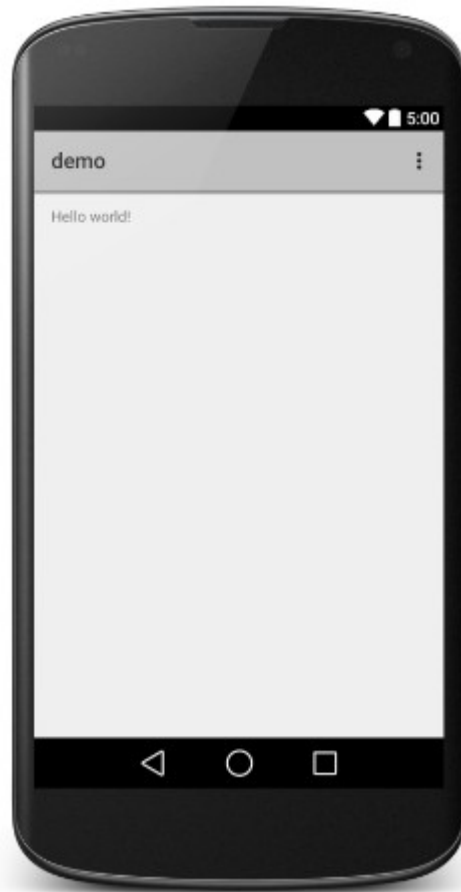
- Select Run ► ➡ ► Run app from the Android Studio menu bar.



Launching Your First Android Application (Contd...)

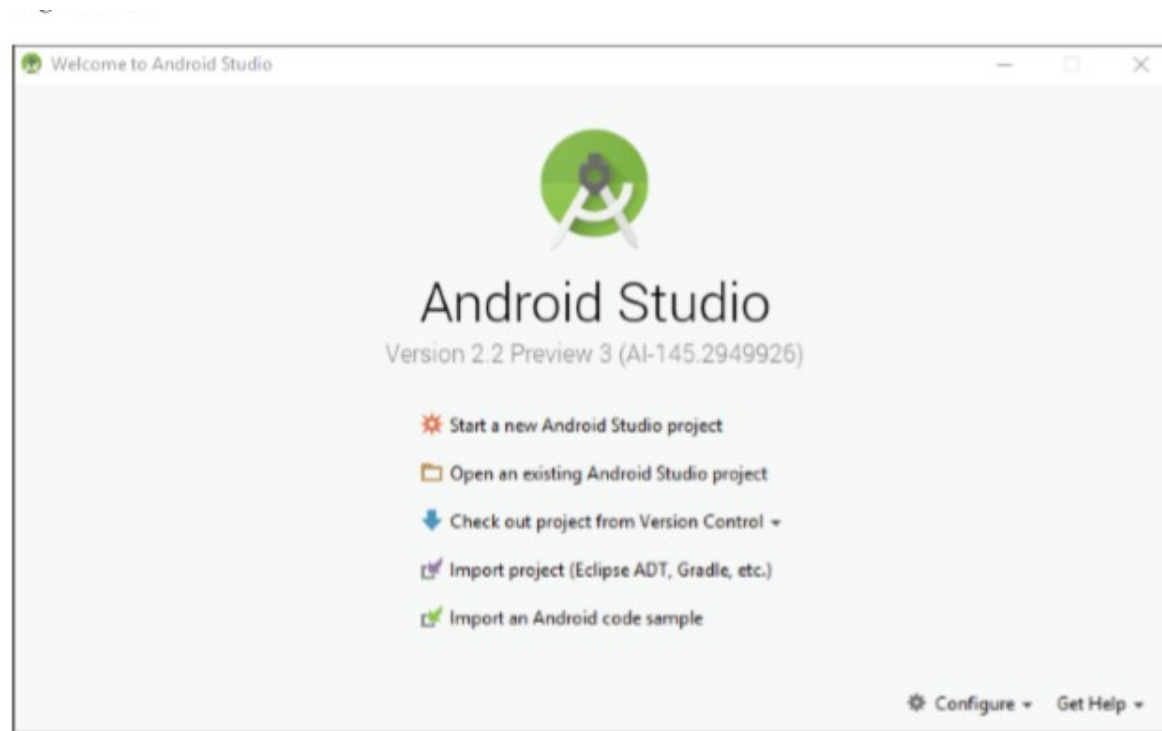
- Select the Nexus 5X API N (feel free to select the Nexus 5x API 18, which is the Jelly Bean emulator), and click Next.
- It can take up to five minutes, and sometimes longer (depending on the hardware specs of your desktop) for the emulator to start and fully load. During this time (the first time you launch the emulator) the application might time out. If a message pops up in Android Studio telling you that the application timed out waiting for the ADB (Android Debugging Bridge) to start, or another similar message, just wait for the emulator to fully load, and then once again select Run ► ⇄ ► Run app from the Android Studio menu bar.

With the emulator fully loaded and started, Android Studio can install Hello World application.



Exploring Android Studio IDE

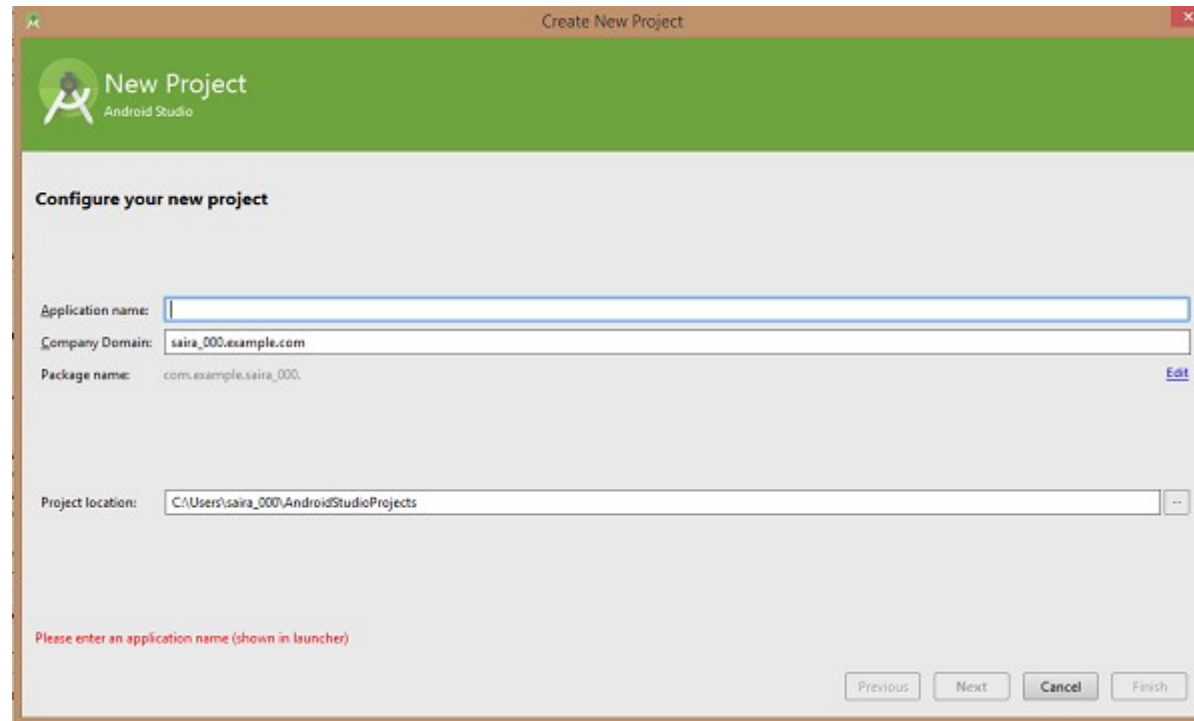
- Now that you have opened Android Studio, you see should a screen that looks like in figure.



Exploring Android Studio IDE (Contd...)

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

Configure New Project

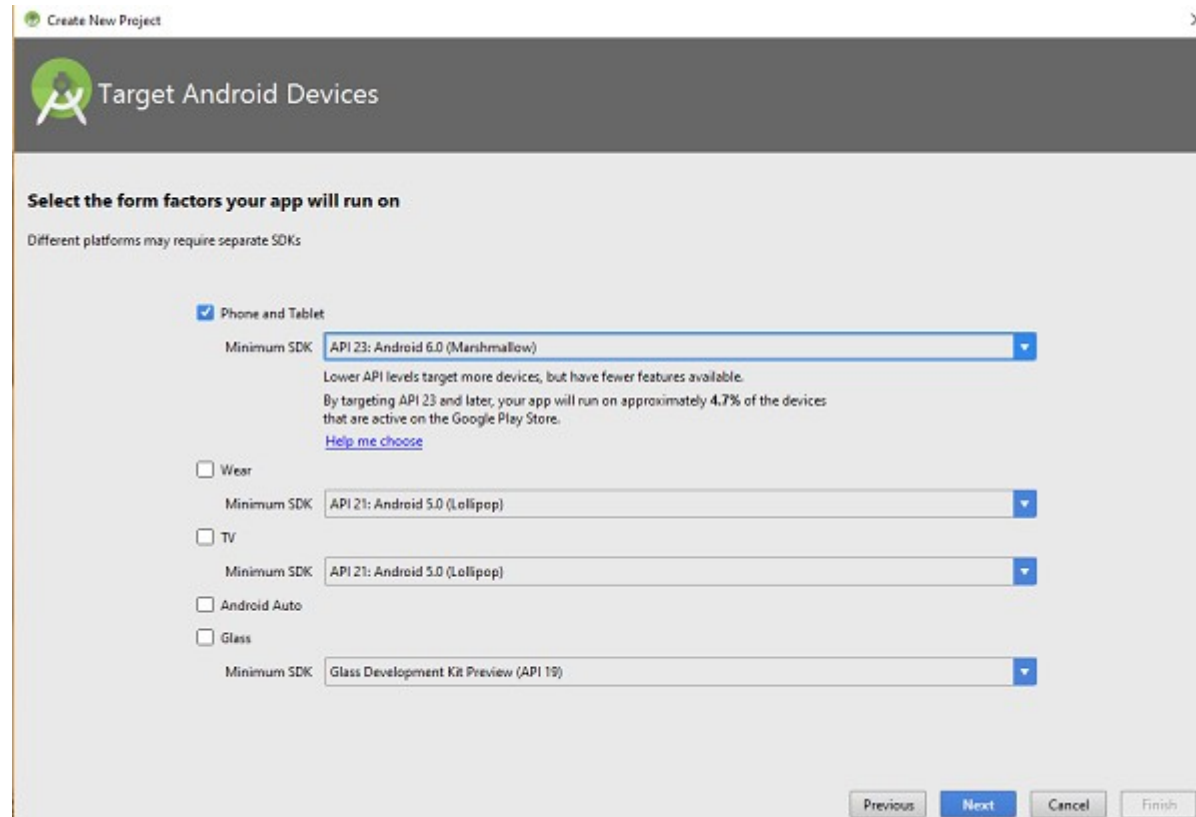


The screenshot shows the 'Create New Project' dialog in Android Studio. The window has a title bar with the text 'Create New Project' and a close button. The main area has a green header with the Android Studio logo and the text 'New Project'. Below the header, the text 'Configure your new project' is displayed. The form contains the following fields:


- Application name:** An empty text field.
- Company Domain:** A text field containing 'saira_000.example.com'.
- Package name:** A text field containing 'com.example.saira_000'. To the right of this field is a blue 'Edit' link.
- Project location:** A text field containing 'C:\Users\saira_000\AndroidStudioProjects'. To the right of this field is a button with two dots '...'.

At the bottom left, there is a red error message: 'Please enter an application name (shown in launcher)'. At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

Selecting the form factors your app will run on



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 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

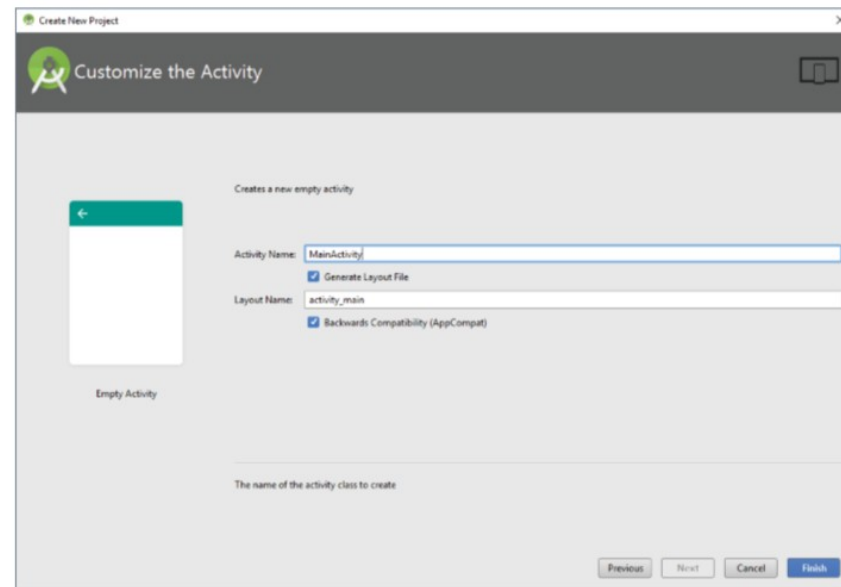
Minimum SDK: Glass Development Kit Preview (API 19)

Previous Next Cancel Finish

Adding an activity to the mobile



Customize the activity screen

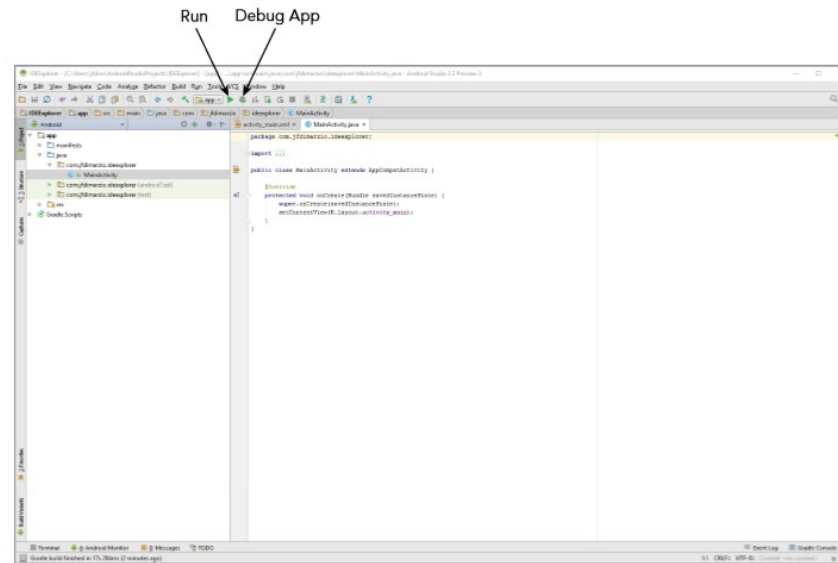


Customize the activity screen

The Customize the Activity screen contains two options, one for naming your activity, and one for naming the main layout (presumably to be used by the main activity).

- It is accepted practice in Android development to name your main activity—that is, the activity that is loaded on startup by your application—as MainActivity. The reason for this is to make it easier to locate the startup code for your application. If anyone else needs to look at or work with your application, they should know that the MainActivity is the starting point. All other activities can be named by their function, for example InputFormActivity or DeleteRecordActivity.
- The layout file follows the “name” naming convention. The startup layout, that is the layout for the screen elements that will be displayed when your application is started by the user, is the activity_main layout. All other layouts should be named according to the activity that they support (activity_input, activity_delete).

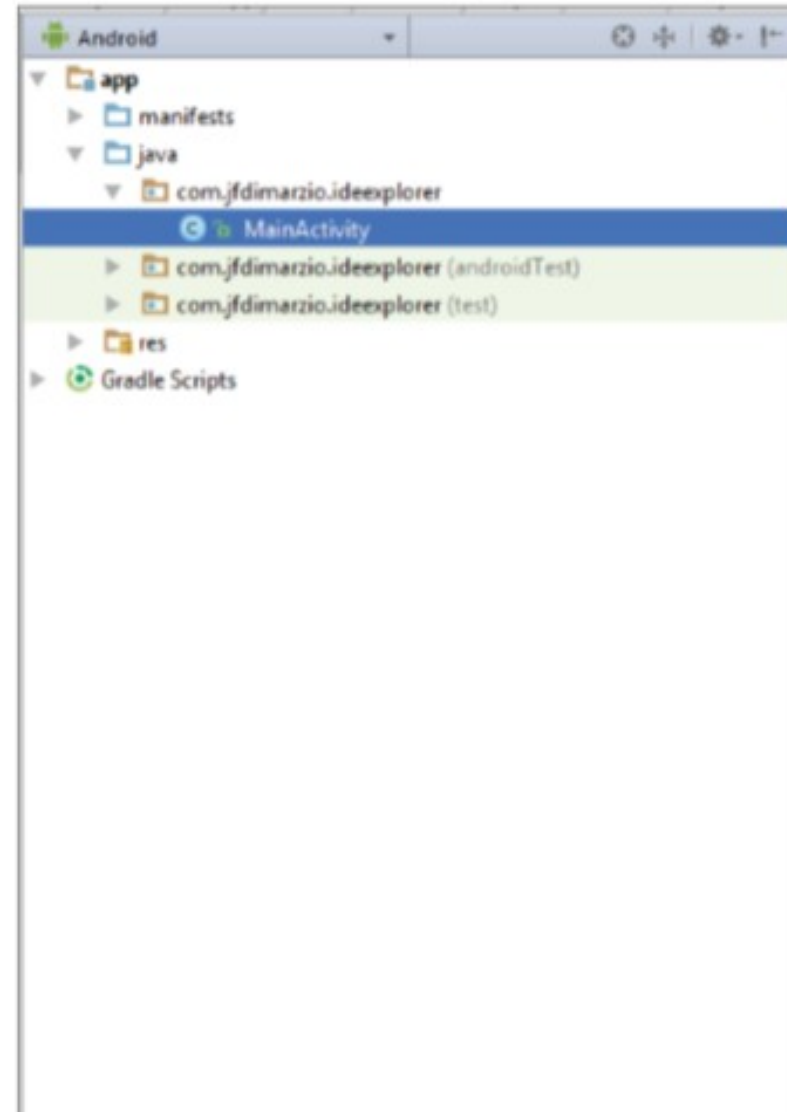
Android Studio IDE



Android Studio IDE

- 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.
- By default, 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. To change the view, click the word Android and use the drop-down list of options to make the change.

Project Window

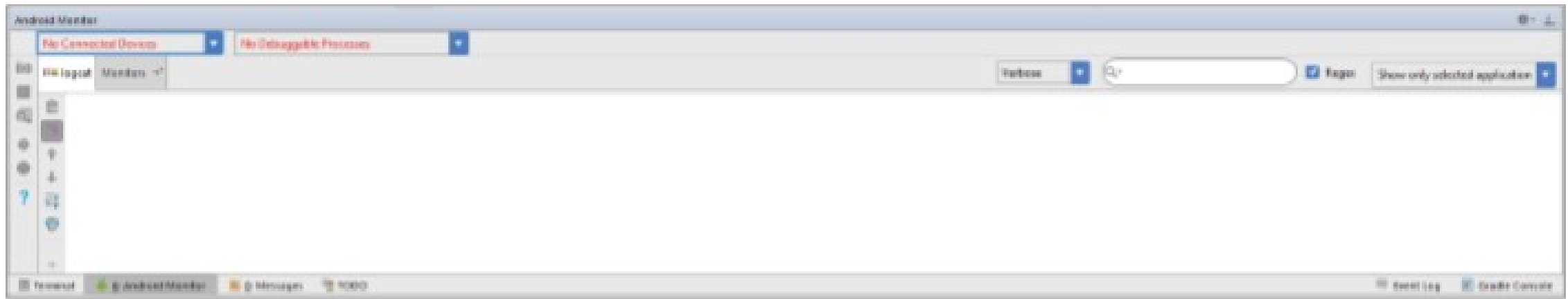


On the right side of the IDE (and taking up the largest area) are the Editor tabs. The Editor tabs are where you write and work with your code files.



Finally, at the bottom of the IDE, you should see a button labelled
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.



Using Code Completion

Code completion is an invaluable tool that shows you contextual options for completing the piece of code that you are trying to write. For example, in the editor tab for the MainActivity.java file, locate the line that reads

```
setContentView(R.layout.activity_main);
```

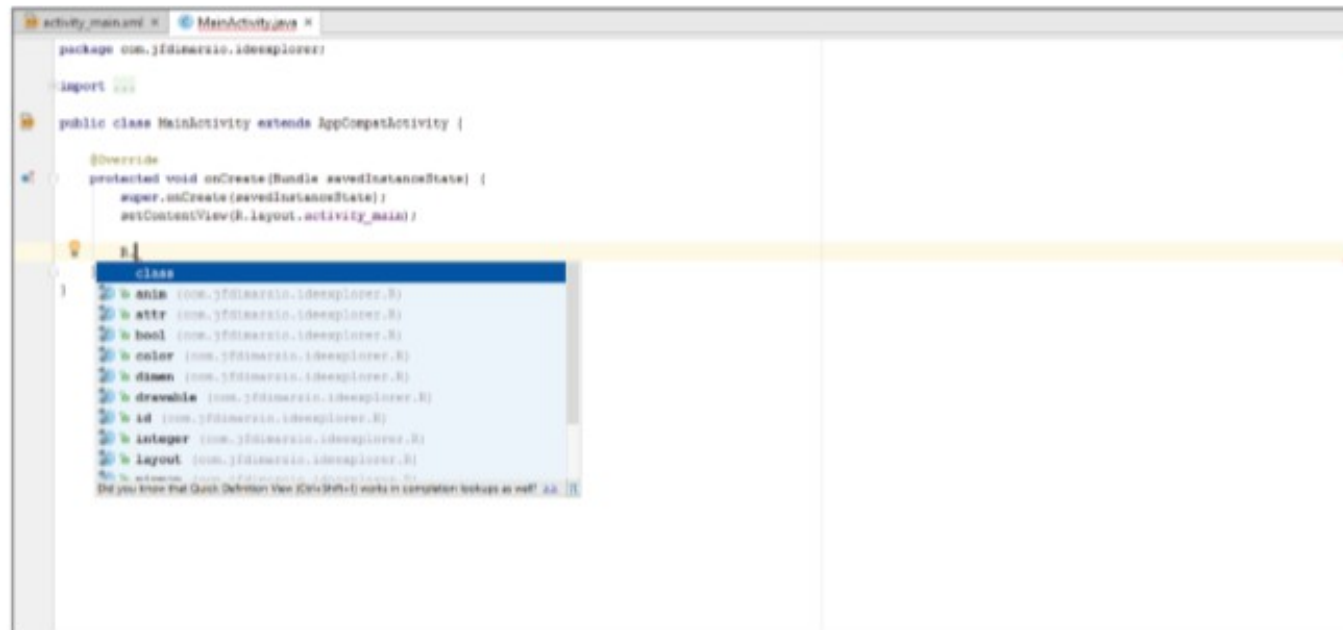
Place your cursor after this line and press the Enter key. On the new line, type the letter R, and then type a period, as shown here:

R.

Android Studio Code Completion should display a list of values that you could use to try to complete the code statement. This is important if you are not entirely sure of the spelling of a method call or of how to identify the different method signatures.

Android Studio Code Completion

You can also use code completion to insert code stubs into your classes. If you are inheriting from a class that has methods you must override, code completion notifies you that there are methods that need to be overridden. With a click, it inserts the stubs of those methods into your application.



Android Studio Code Completion

For example, if you were to attempt to create a variable of a type that belongs to a package that you have not imported, Android Studio recognizes this and underlines the type with a red squiggle. Set the cursor to that line and press `Alt+Enter` to automatically import the package into a using statement at the top of your code file.

Debugging Your Application

Breakpoints allow you to pause the execution of your code at specific locations and see what is going on (or what is going wrong).

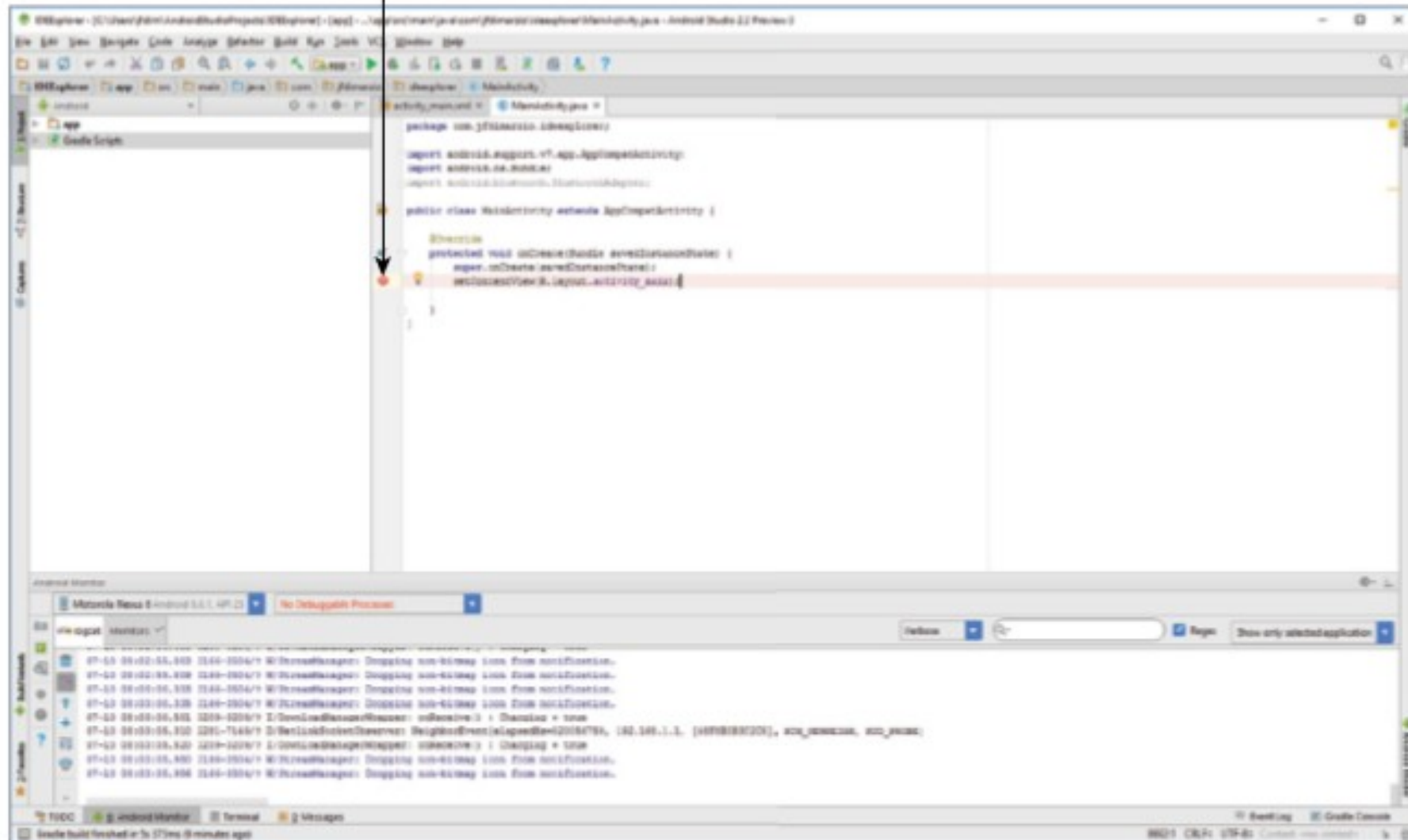
Setting breakpoints

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.

Setting breakpoints

A breakpoint is set for this line.



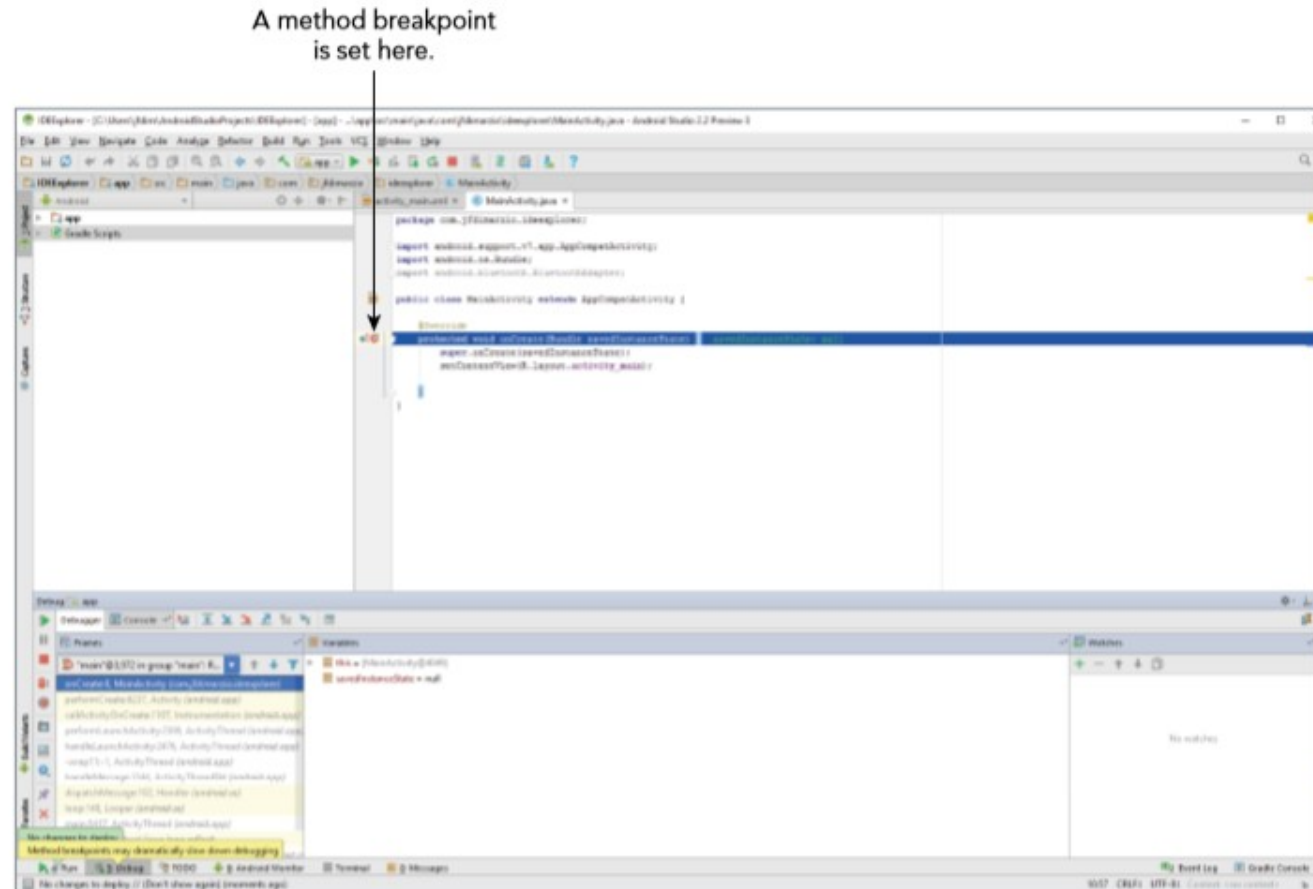
Removing Breakpoint

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.

Note: Android Studio only pauses execution at breakpoints when you debug your application—not when you run it. This means you must use the green arrow with the bug behind it (or select Run ► ⇄ ► Debug ‘app’, or press Shift+F9).

Setting breakpoints for a method

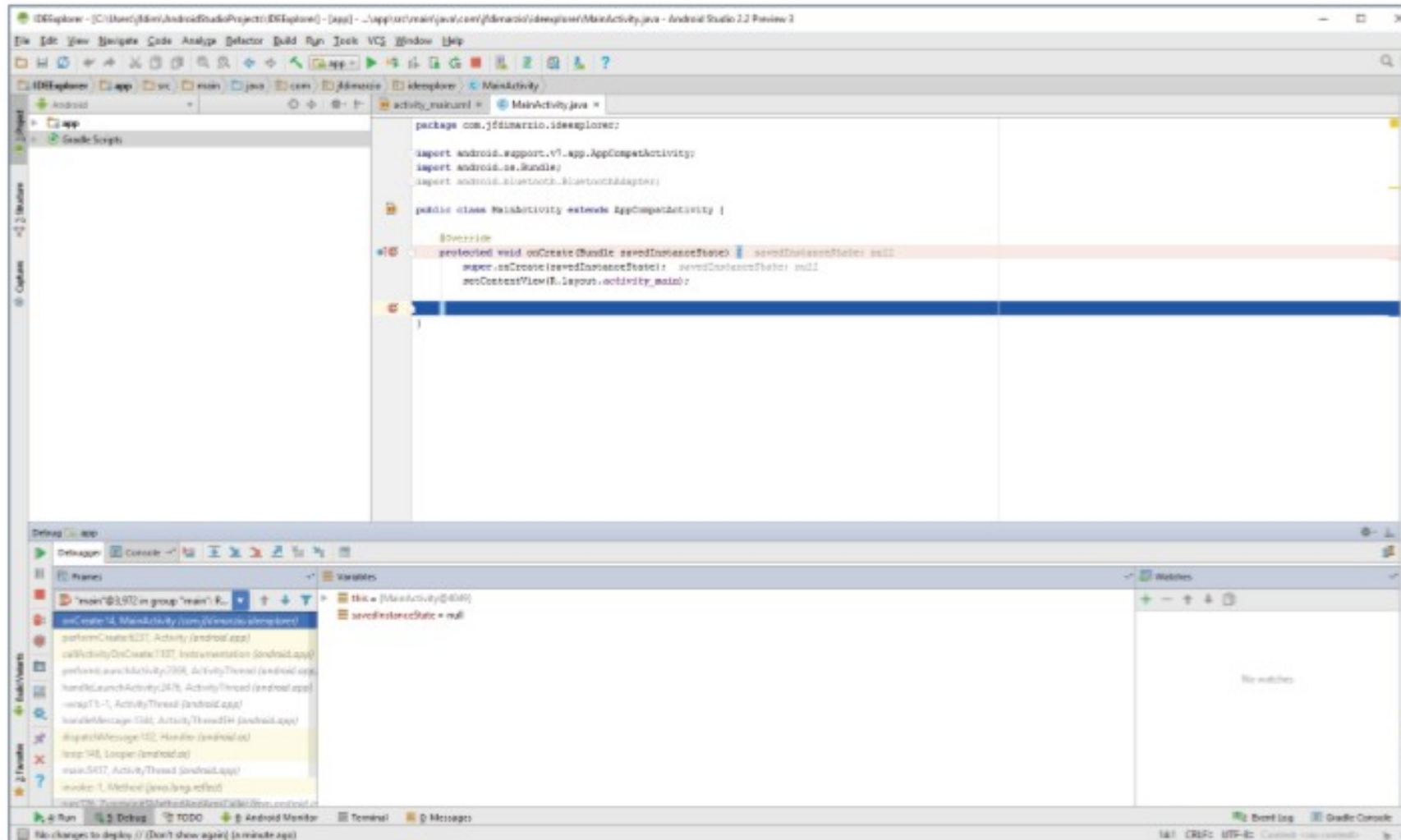
- Set a method breakpoint by selecting Run ► ⚙ ► Toggle Method Breakpoint. A method breakpoint is represented by a red circle containing four dots placed at the method signature



Setting breakpoints for a method

- Android Studio issues a warning that method breakpoints can dramatically slow down debugging. This is because method breakpoints do more than simple breakpoints in their default state. By default, method breakpoints are set apart from simple breakpoints.
- Android Studio pauses execution when the method is hit, and it also automatically sets a corresponding breakpoint and pauses at the end of the method

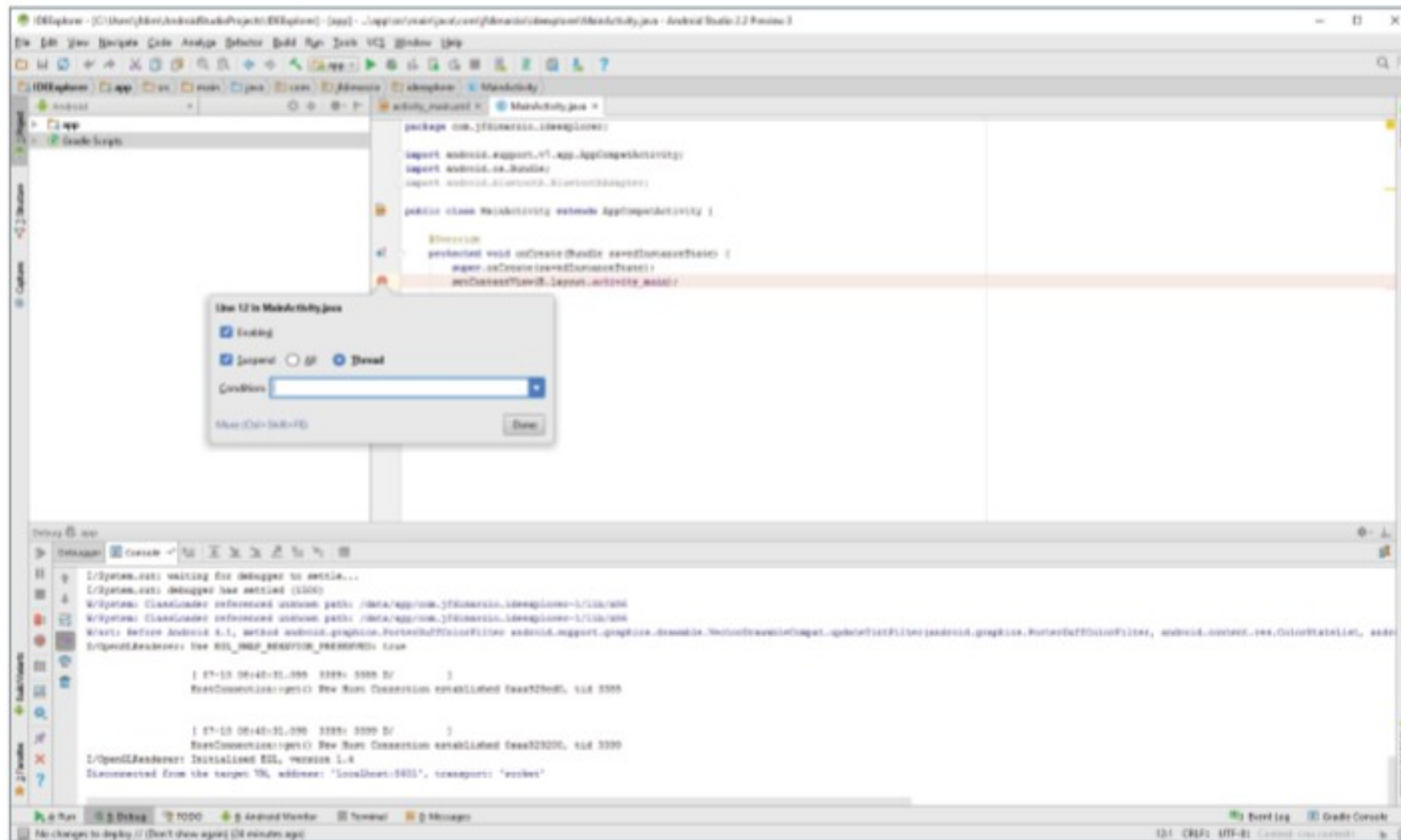
Click in the margin of the editor tab at the signature of a method to set a simple breakpoint there



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.

Temporary Breakpoints



Temporary Breakpoints

- 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

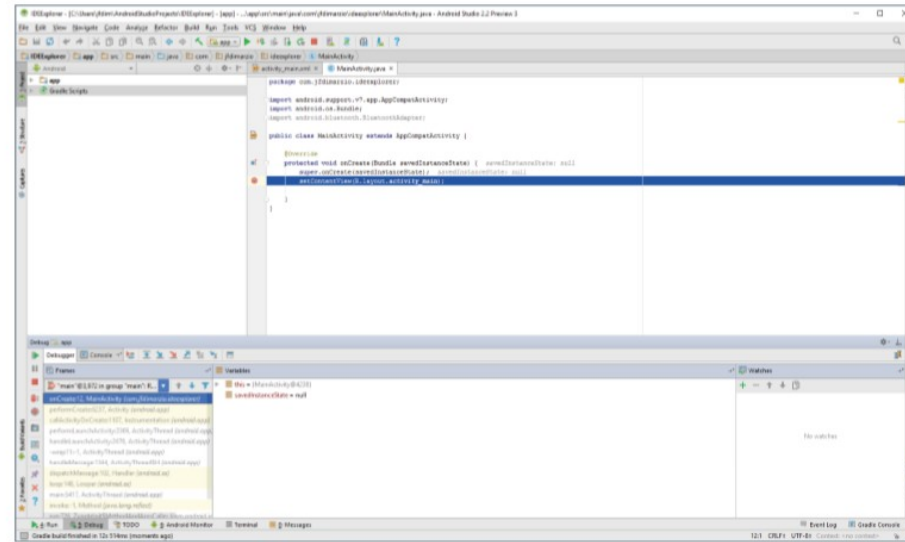
`foo == true`

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

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.

Navigating paused Code



Once a breakpoint has been hit, the debug window opens at the bottom of Android Studio, which contains many of the tools used to navigate around code .

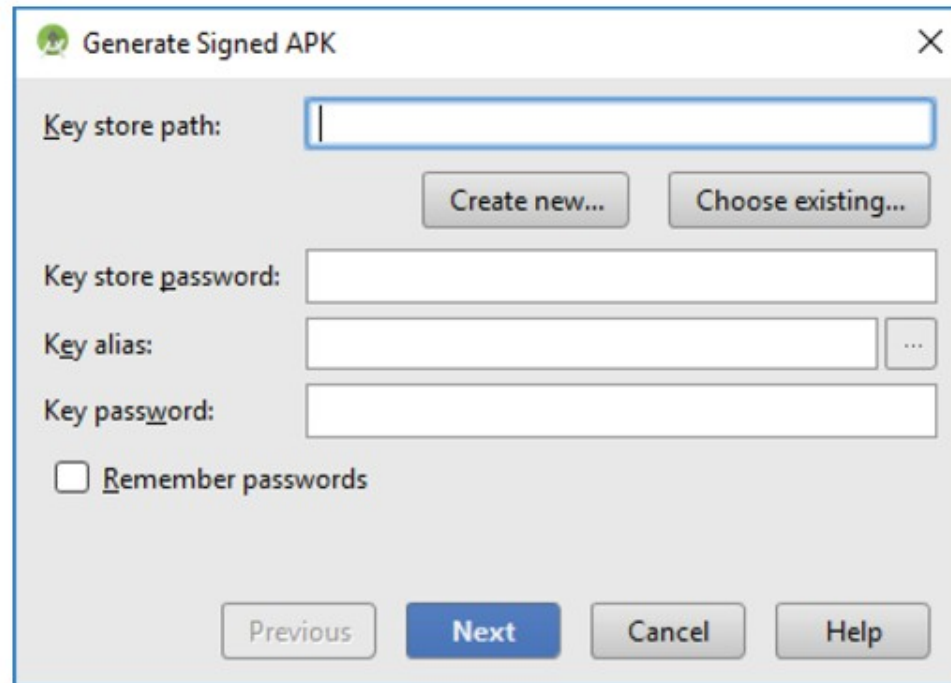


Navigating paused Code

- 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. Therefore, 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

- 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



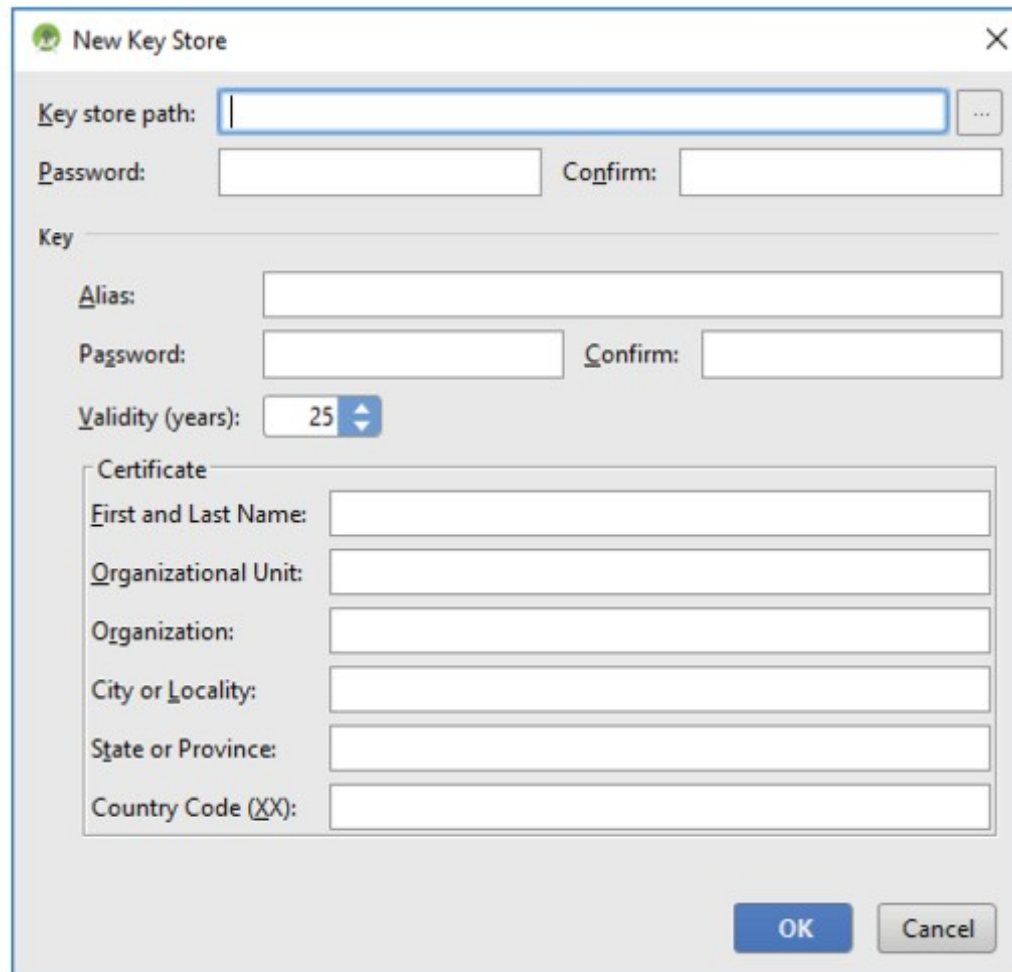
The screenshot shows the 'Generate Signed APK' dialog box. It has a title bar with a green icon and a close button. The main area contains four text input fields: 'Key store path:', 'Key store password:', 'Key alias:', and 'Key password:'. Below the 'Key store path:' field are two buttons: 'Create new...' and 'Choose existing...'. Below the 'Key alias:' field is a button with three dots. At the bottom left is a checkbox labeled 'Remember passwords'. At the bottom right are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Help'.

Publishing Your Application

- 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
- 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.

Publishing Your Application



A screenshot of the 'New Key Store' dialog box in a software application. The dialog has a title bar with a green icon and a close button. It contains several input fields for configuring a new key store. The 'Key store path' field is highlighted with a blue border. Below it are 'Password' and 'Confirm' fields. A 'Key' section contains 'Alias', 'Password', and 'Confirm' fields. A 'Validity (years)' spinner is set to 25. A 'Certificate' section contains fields for 'First and Last Name', 'Organizational Unit', 'Organization', 'City or Locality', 'State or Province', and 'Country Code (XX)'. At the bottom are 'OK' and 'Cancel' buttons.

New Key Store

Key store path: ...

Password: Confirm:

Key

Alias:

Password: Confirm:

Validity (years): 25

Certificate

First and Last Name:

Organizational Unit:

Organization:

City or Locality:

State or Province:

Country Code (XX):

OK Cancel

Publishing Your Application

- Click OK to return to the Generate Signed APK window.
- 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/>.