Android Studio

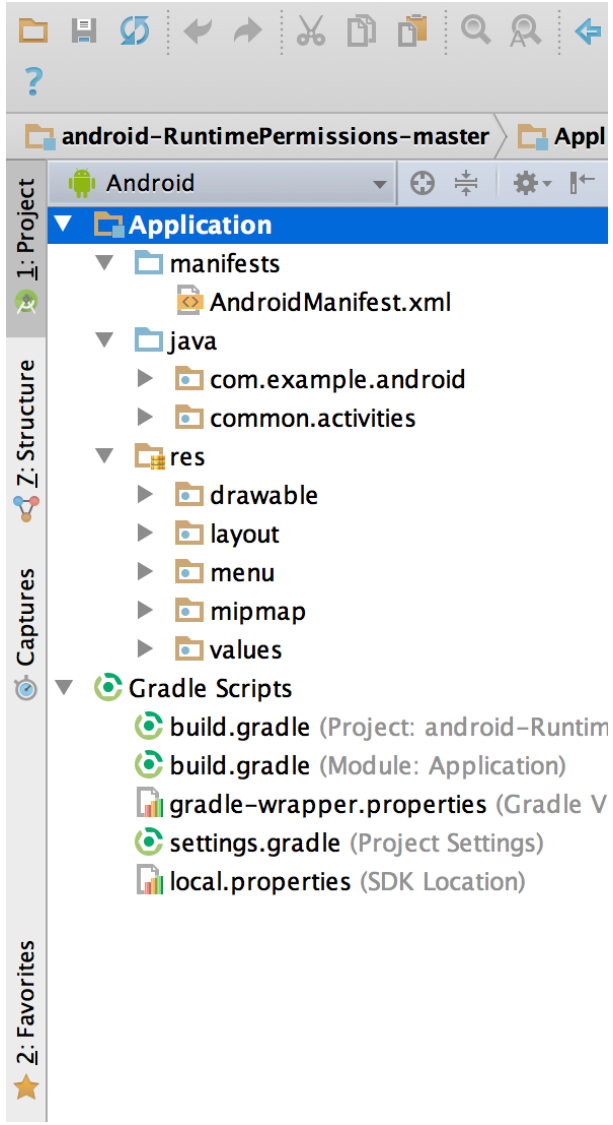
Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Project structure

1. Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:
2. Android app modules
3. Library modules
4. Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.



All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

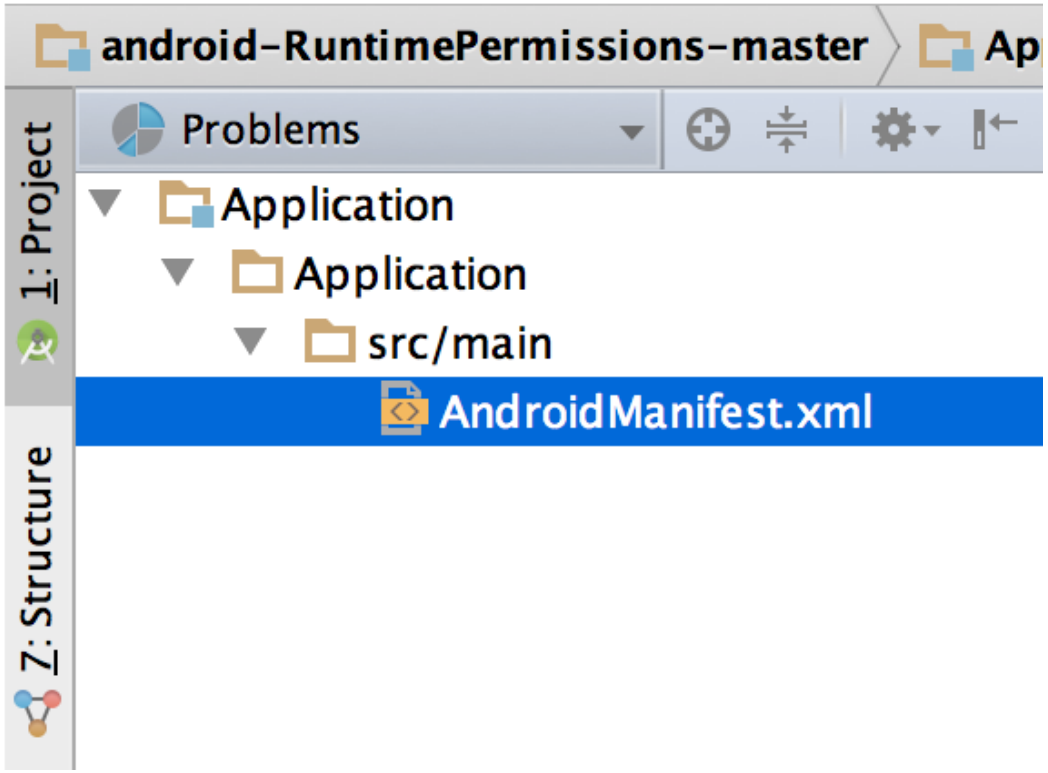
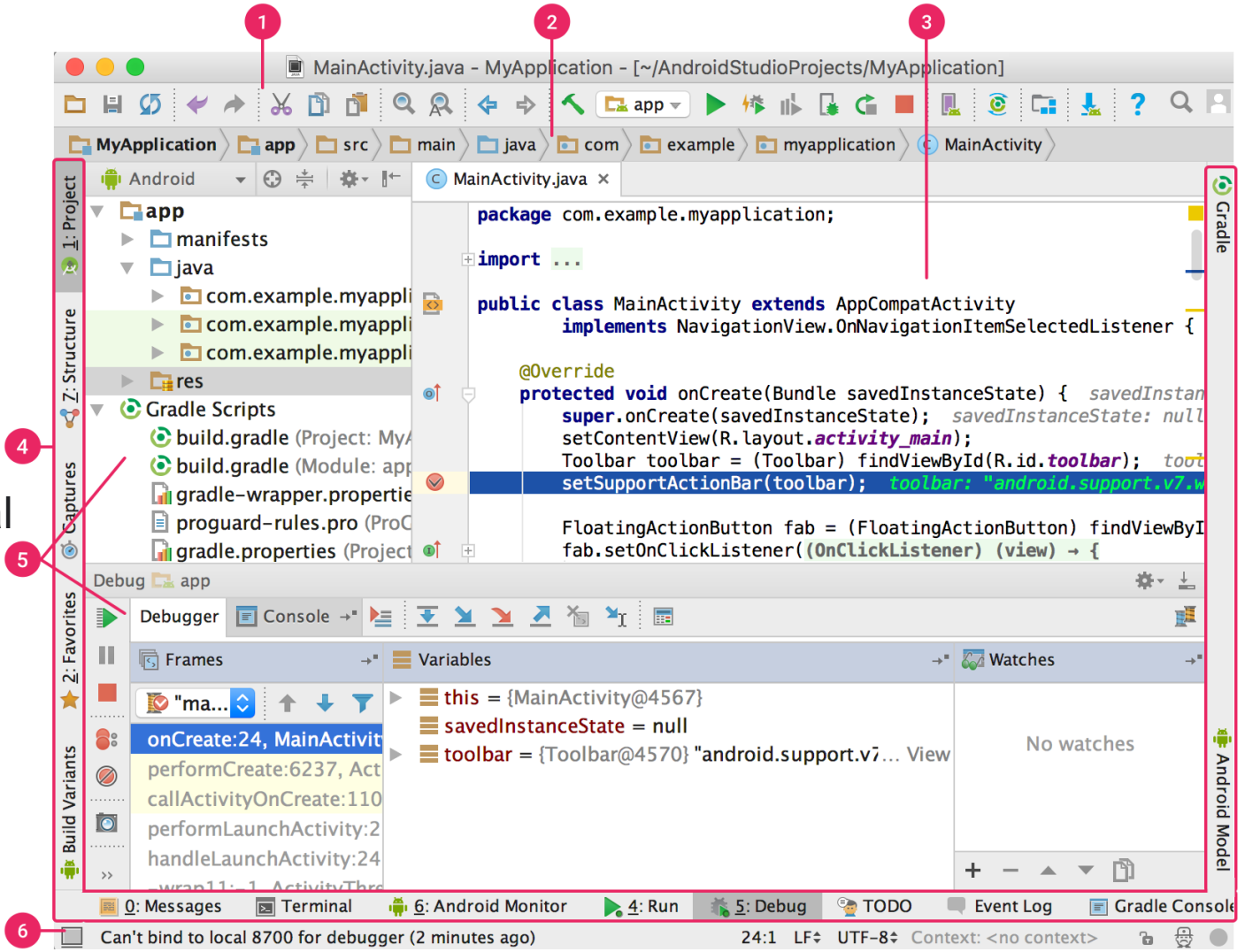


Figure 2. The project files in Problems view, showing a layout file with a problem.

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

Figure 3. The Android Studio main window.



Tool windows:

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.
- To show or hide the entire tool window bar, click the window icon in the bottom left-hand corner of the Android Studio window.
- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

Table 1. Keyboard shortcuts for some useful tool windows.

Tool window	Windows and Linux	Mac
Project	Alt+1	Command+1
Version Control	Alt+9	Command+9
Run	Shift+F10	Control+R
Debug	Shift+F9	Control+D
Logcat	Alt+6	Command+6
Return to Editor	Esc	Esc
Hide All Tool Windows	Control+Shift+F12	Command+Shift+F12

If you want to hide all toolbars, tool windows, and editor tabs, click **View > Enter Distraction Free Mode**. This enables *Distraction Free Mode*. To exit Distraction Free Mode, click **View > Exit Distraction Free Mode**.

You can use *Speed Search* to search and filter within most tool windows in Android Studio. To use Speed Search, select the tool window and then type your search query.
For more tips, see Keyboard shortcuts.

Code completion:

Android Studio has three types of code completion, which you can access using keyboard shortcuts.
Table 2. Keyboard shortcuts for code completion.

Type	Description	Windows and Linux	Mac
Basic Completion	Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.	Control+Spac e	Control+Space
Smart Completion	Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains.	Control+Shift +Space	Control+Shift+S pace
Statement Completion	Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.	Control+Shift +Enter	Shift+Command +Enter

Navigation

Here are some tips to help you move around Android Studio.

- Switch between your recently accessed files using the *Recent Files* action.

  Press **Control+E** (**Command+E** on a Mac) to bring up the Recent Files action. By default, the last accessed file is selected. You can also access any tool window through the left column in this action.
- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12** (**Command+F12** on a Mac). Using this action, you can quickly navigate to any part of your current file.

- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N** (**Command+O** on a Mac). Navigate to Class supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice in a row, it shows you the results out of the project classes.
- Navigate to a file or folder using the *Navigate to File* action. Bring up the Navigate to File action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac). To search for folders rather than files, add a / at the end of your expression.
- Navigate to a method or field by name using the *Navigate to Symbol* action. Bring up the Navigate to Symbol action by pressing **Control+Shift+Alt+N** (**Command+Option+O** on a Mac).
- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7** (**Option+F7** on a Mac)

Style and formatting

•As you edit, Android Studio automatically applies formatting and styles as specified in your code style settings. You can customize the code style settings by programming language, including specifying conventions for tabs and indents, spaces, wrapping and braces, and blank lines. To customize your code style settings, click **File > Settings > Editor > Code Style** (**Android Studio > Preferences > Editor > Code Style** on a Mac.)
•Although the IDE automatically applies formatting as you work, you can also explicitly call the *Reformat Code* action by
pressing **Control+Alt+L** (**Opt+Command+L** on a Mac), or auto-indent all lines by pressing **Control+Alt+I** (**Control+Option+I** on a Mac).

Figure 4. Code before formatting.

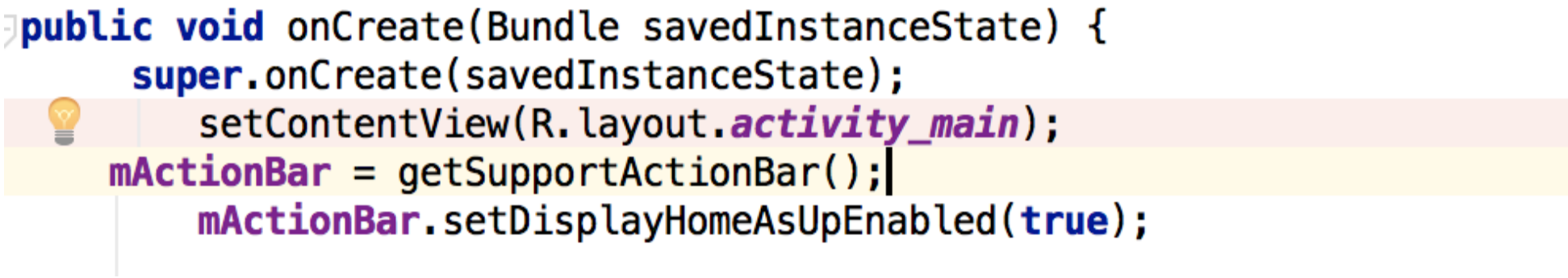
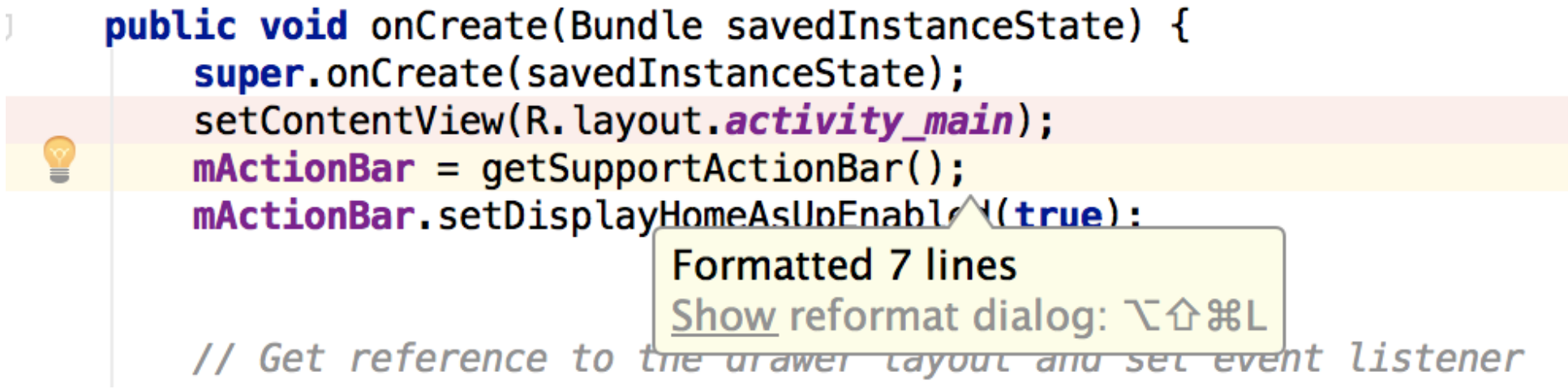


Figure 5. Code after formatting.



Version control basics:

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.
After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations:

- From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
- From the drop-down menu, select a version control system to associate with the project root, and then click **OK**.

The VCS menu now displays a number of version control options based on the system you selected.

Gradle build system:

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across sourcesets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

To learn more about the build system and how to configure, see Configure your build.

Build variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

For more information about configuring build variants, see Configure build variants.

Multiple APK support

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the hdpi and mdpi screen densities, while still considering them a single variant and allowing them to share test APK, javac, dx, and ProGuard settings.

For more information about multiple APK support, read Build multiple APKs.

Resource shrinking

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using Google Play services to access Google Drive functionality, and you are not currently using Google Sign-In, then resource shrinking can remove the various drawable assets for theSignInButton buttons.

Managing dependencies

Dependencies for your project are specified by name in the build.gradle file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle file. Android Studio configures projects to use the Maven Central Repository by default. (This configuration is included in the top-level build file for the project.) For more information about configuring dependencies, read Add build dependencies.

Debug and profile tools:

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline debugging:

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values

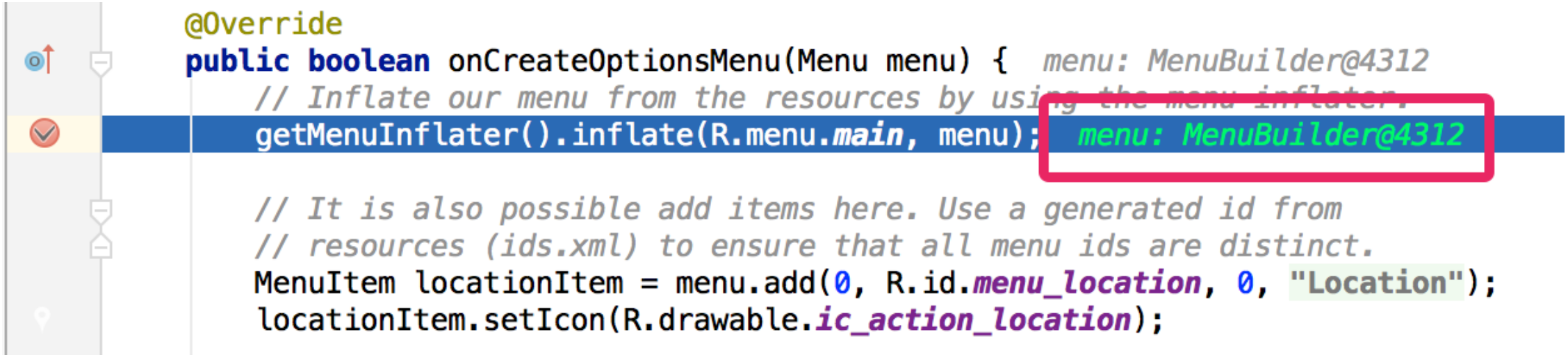


Figure 6. An inline variable value.
To enable inline debugging, in the **Debug** window, click **Settings** and select the checkbox for **Show Values Inline**.

Performance profilers

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler** tab.
For more information about performance profilers, see Performance profiling tools.

Heap dump

When you're profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app's performance and memory use by adjusting the method calls related to those actions.
For information about tracking and analyzing allocations, see Inspect the heap and allocations.

Data file access

The Android SDK tools, such as Systrace, and logcat, generate performance and debugging data for detailed app analysis.
To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to the standard Investigate your RAM usage file format.

Whenever you compile your program, Android Studio automatically runs configured Lint and other IDE inspections to help you easily identify and correct problems with the structural quality of your code.
The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

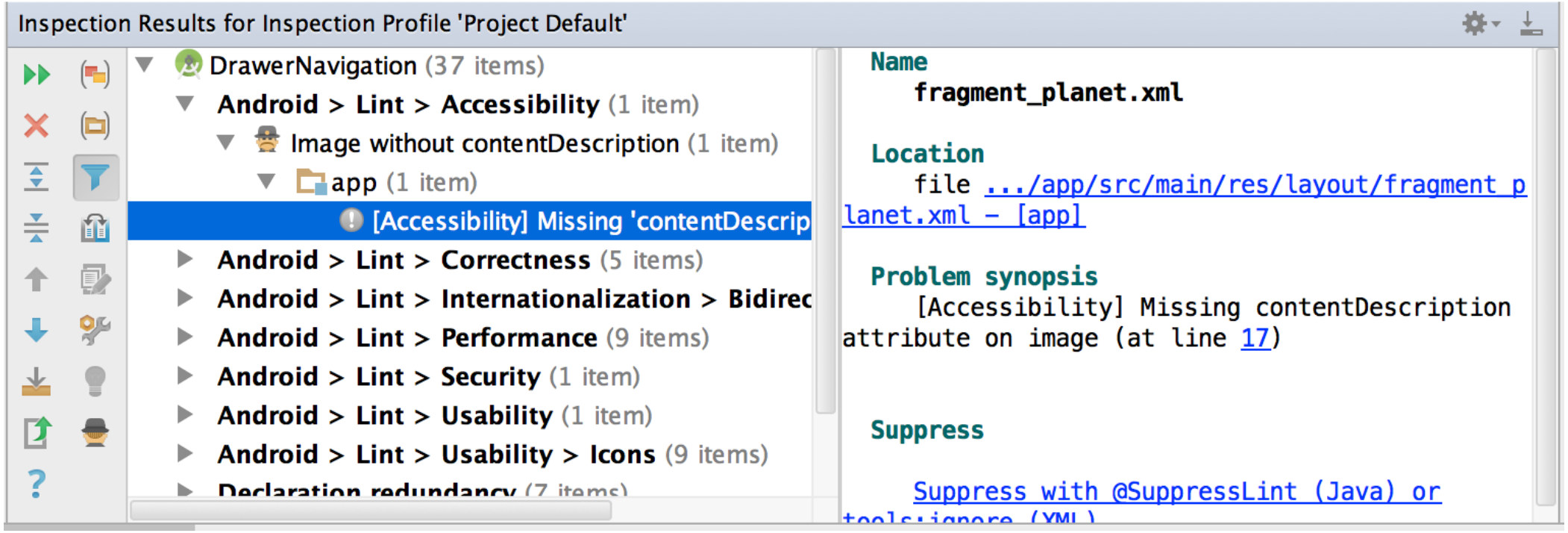


Figure 7. The results of a Lint inspection in Android Studio.
In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.
For more information, see Improve your code with lint checks.

Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.
For more details about Android annotations, see Improve code inspection with annotations.

Log messages

When you build and run your app with Android Studio, you can view adb output and device log messages in the **Logcat** window.

Performance profiling

If you want to profile your app's CPU, memory, and network performance, open the Android Profiler, by clicking **View > Tool Windows > Android Profiler**.

Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.
For more details about Android annotations, see Improve code inspection with annotations.

Log messages

When you build and run your app with Android Studio, you can view adb output and device log messages in the **Logcat** window.

Performance profiling

If you want to profile your app's CPU, memory, and network performance, open the Android Profiler, by clicking **View > Tool Windows > Android Profiler**.

Projects overview

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

If you don't have a project opened, Android Studio shows the Welcome screen, where you can create a new project by clicking **Start a new Android Studio project**.

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

You then see the **Create New Project** wizard, which lets you choose the type of project you want to create and populates with code and resources to get you started. This page guides you through creating a new project using the **Create New Project** wizard.

Choose your project:

In the **Choose your project** screen that appears, you can select the type of project you want to create from categories of device form factors, which are shown as tabs near the top of the wizard. For example, figure 1 shows a project with a basic Android Activity for a phone and tablet selected.

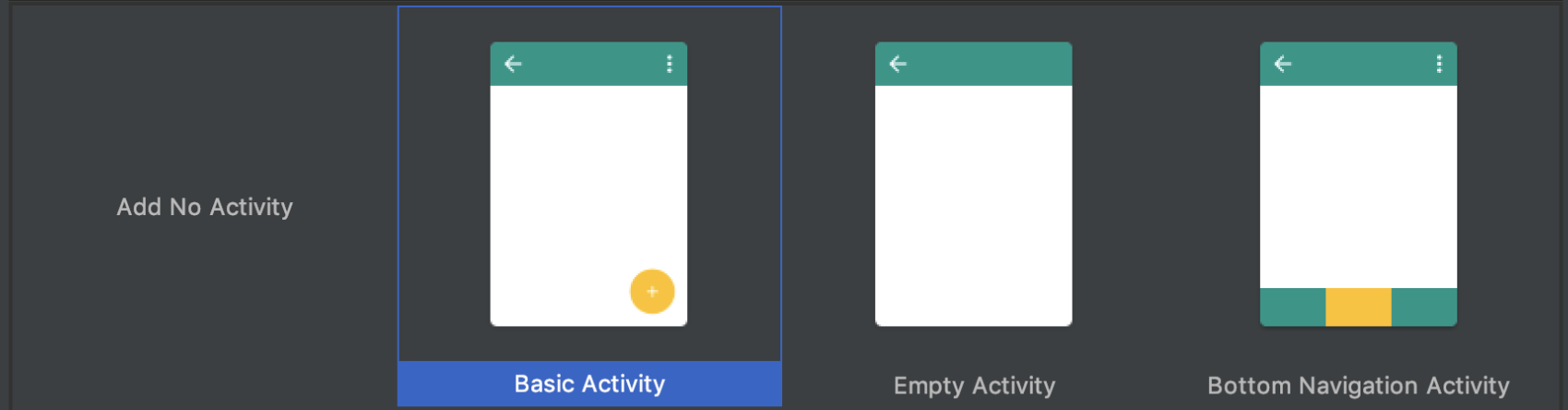
Figure 1. In the first screen of the wizard, choose the type of project you want to create.

By selecting the type of project you want to create, Android Studio can include sample code and resources to help you get started.

After you make a selection, click Next.

Choose your project

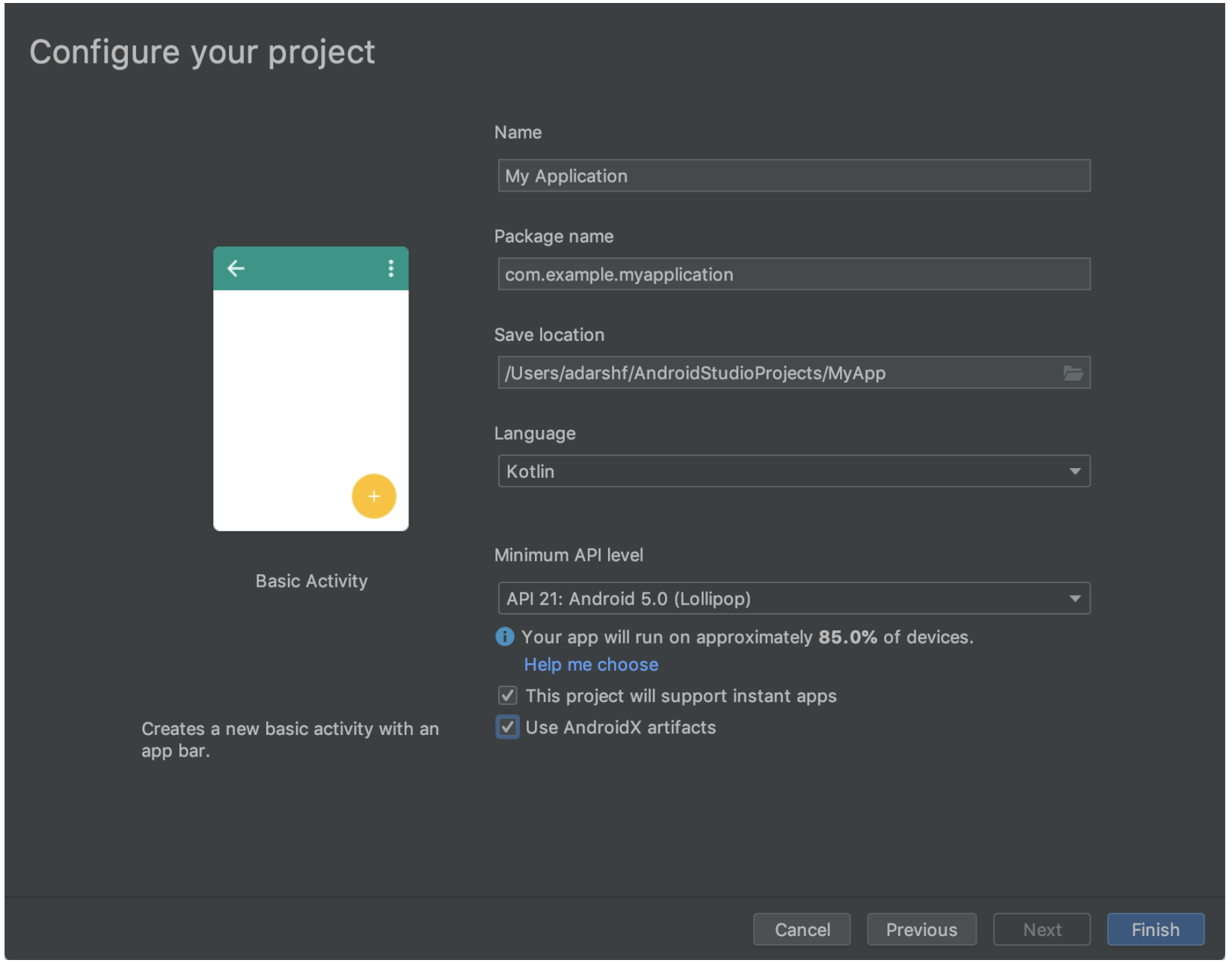
Phone and Tablet | Wear OS | TV | Android Auto | Android Things



Basic Activity

Creates a new basic activity with an app bar.

Cancel | Previous | Next | Finish



Configure your project:

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

Figure 2. Configure your new project with a few settings.

1.Specify the **Name** of your project.

2.Specify the **Package name**. By default, this package name also becomes your application ID, which you can change later.

3.Specify the **Save location** where you want to locally store your project.

4.Select the **Language** you want Android Studio to use when creating sample code for your new project. Keep in mind, you are *not* limited to using only that language in the project.

5.Select the **Minimum API level** you want your app to support. When you select a lower API level, your app can't use as many modern Android APIs. However, a larger percentage of Android devices are able to run your app. The opposite is true when selecting a higher API level. If you want to see more data to help you decide, click **Help me choose**.

6.If you want your project to use AndroidX libraries by default, which are improved replacements of the Android Support libraries, check the box next to **Use AndroidX artifacts**. To learn more, read the AndroidX overview.

7.When you're ready to create your project, click **Finish**.

Import an existing project

To import an existing, local project into Android Studio, proceed as follows:

1. Click **File** > **New** > **Import Project**.
2. In the window that appears, navigate to the root directory of the project you want to import.
3. Click **OK**.

Android Studio then opens the project in a new IDE window and indexes its contents.

If you are importing a project from version control, use the **File** > **New** > **Project from Version Control** menu.

For more information about importing projects from version control, read IntelliJ's VCS-Specific Procedures.

Projects overview

A *project* in Android Studio contains everything that defines your workspace for an app, from source code and assets, to test code and build configurations. When you start a new project, Android Studio creates the necessary structure for all your files and makes them visible in the **Project** window on the left side of the IDE (click **View > Tool Windows > Project**). This page provides an overview of the key components inside your project.

Modules

A *module* is a collection of source files and build settings that allow you to divide your project into discrete units of functionality. Your project can have one or many modules, and one module may use another module as a dependency. You can independently build, test, and debug each module. Additional modules are often useful when creating code libraries within your own project or when you want to create different sets of code and resources for different device types, such as phones and wearables, but keep all the files scoped within the same project and share some code.
Add a new module to your project by clicking **File > New > New Module**.
Android Studio offers a few distinct types of module:

Android app module

Provides a container for your app's source code, resource files, and app level settings such as the module-level build file and Android Manifest file. When you create a new project, the default module name is "app".

In the **Create New Module** window, Android Studio offers the following types of app modules:

- Phone & Tablet Module
- Wear OS Module
- Android TV Module
- Glass Module

Each provides essential files and some code templates that are appropriate for the corresponding app or device type.

For more information on adding a module, read Add a Module for a New Device.

Feature module

Represents a modularized feature of your app that can take advantage of *Play Feature Delivery*. For example, with feature modules, you can provide your users with certain features of your app on-demand or as instant experiences through Google Play Instant.

To learn more, read Add support for Play Feature Delivery.

Library module

Provides a container for your reusable code, which you can use as a dependency in other app modules or import into other projects. Structurally, a library module is the same as an app module, but when built, it creates a code archive file instead of an APK, so it can't be installed on a device.

In the **Create New Module** window, Android Studio offers the following library modules:

- Android Library: This type of library can contain all file types supported in an Android project, including source code, resources, and manifest files. The build result is an Android Archive (AAR) file that you can add as a dependency for your Android app modules.
- Java Library: This type of library can contain only Java source files. The build result is an Java Archive (JAR) file that you can add as a dependency for your Android app modules or other Java projects.

Google Cloud module:

Provides a container for your Google Cloud backend code. This module has the required code and dependencies for a Java App Engine backend that uses simple HTTP, Cloud Endpoints, and Cloud Messaging to connect to your app. You can develop your backend to provide cloud services your app needs.

Using Android Studio to develop your Google Cloud module lets you manage app code and backend code in the same project. You can also run and test your backend code locally, and use Android Studio to deploy your Google Cloud module.

For more information on running and deploying a Google Cloud module, see Running, Testing, and Deploying the Backend.

Some people also refer to modules as sub-projects and that's okay, because Gradle also refers to modules as projects. For example, when you create a library module and want to add it as a dependency to your Android app module, you must declare it as follows:

```
dependencies {
   implementation(project(":my-library-module"))
}
```

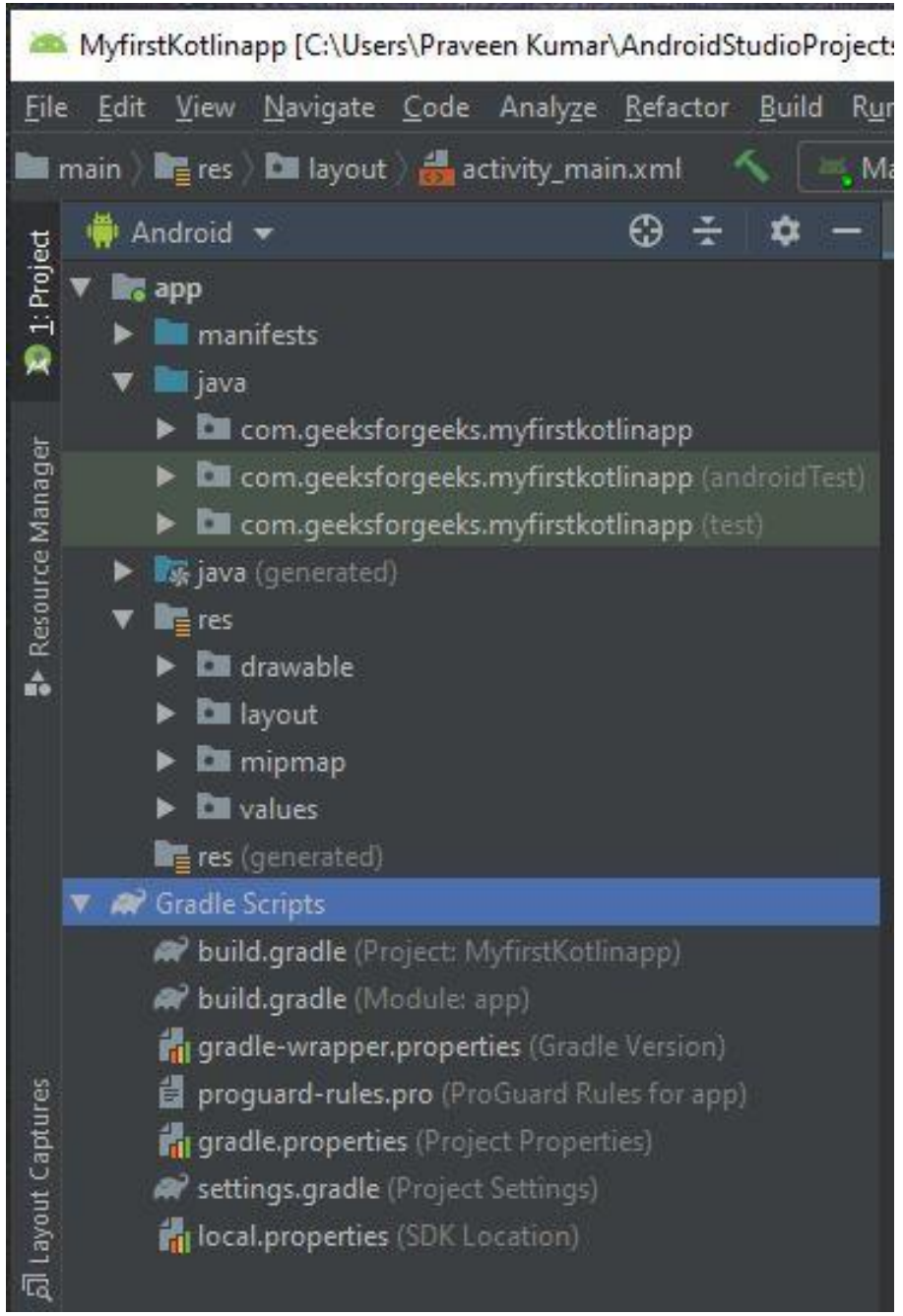
Android Project folder Structure:

Android Studio is the official IDE (Integrated Development Environment) developed by JetBrains community which is freely provided by Google for android app development.
After completing the setup of Android Architecture we can create android application in the studio. We need to create new project for each sample application and we should understand the folder structure. It looks like this:

Android Project folder Structure:

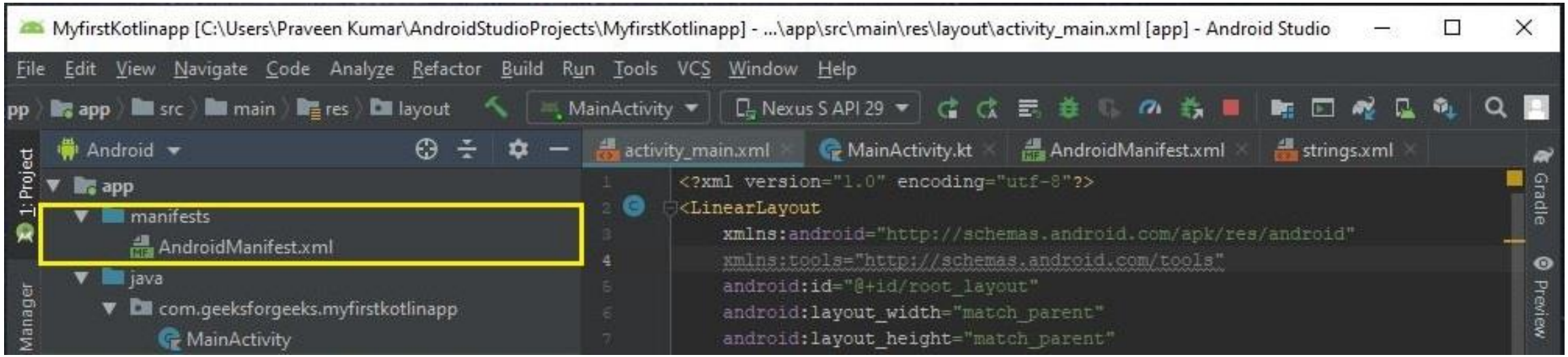
The android project contains different types of app modules, source code files, and resource files. We will explore all the folders and files in the android app.

1.Manifests Folder
2.Java Folder
3.res (Resources) Folder
    1.Drawable Folder
    2.Layout Folder
    3.Mipmap Folder
    4.Values Folder
4.Gradle Scripts



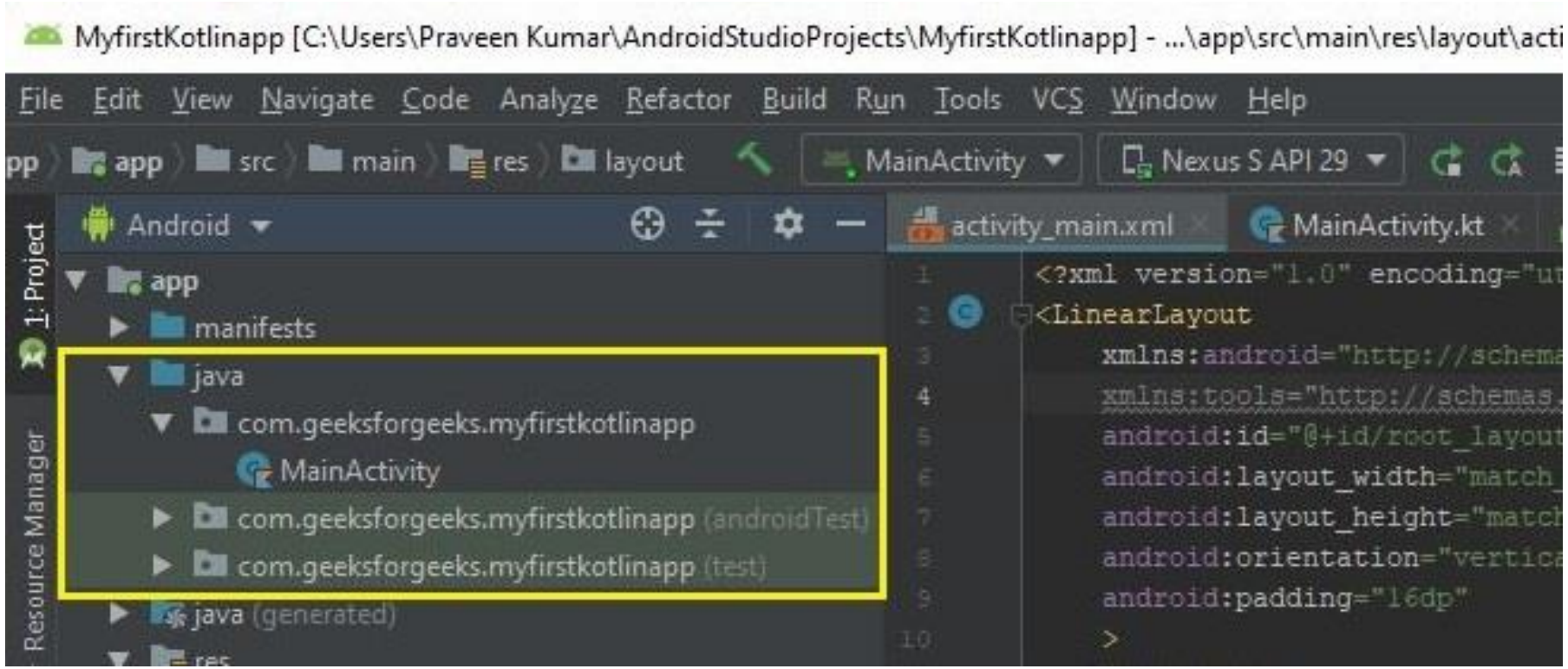
Manifests Folder

Manifests folder contains **AndroidManifest.xml** for our creating the android application. This file contains information about our application such as android version, metadata, states package for Kotlin file and other application components. It acts as an intermediator between android OS and our application.
Following is the mainfests folder structure in android application.



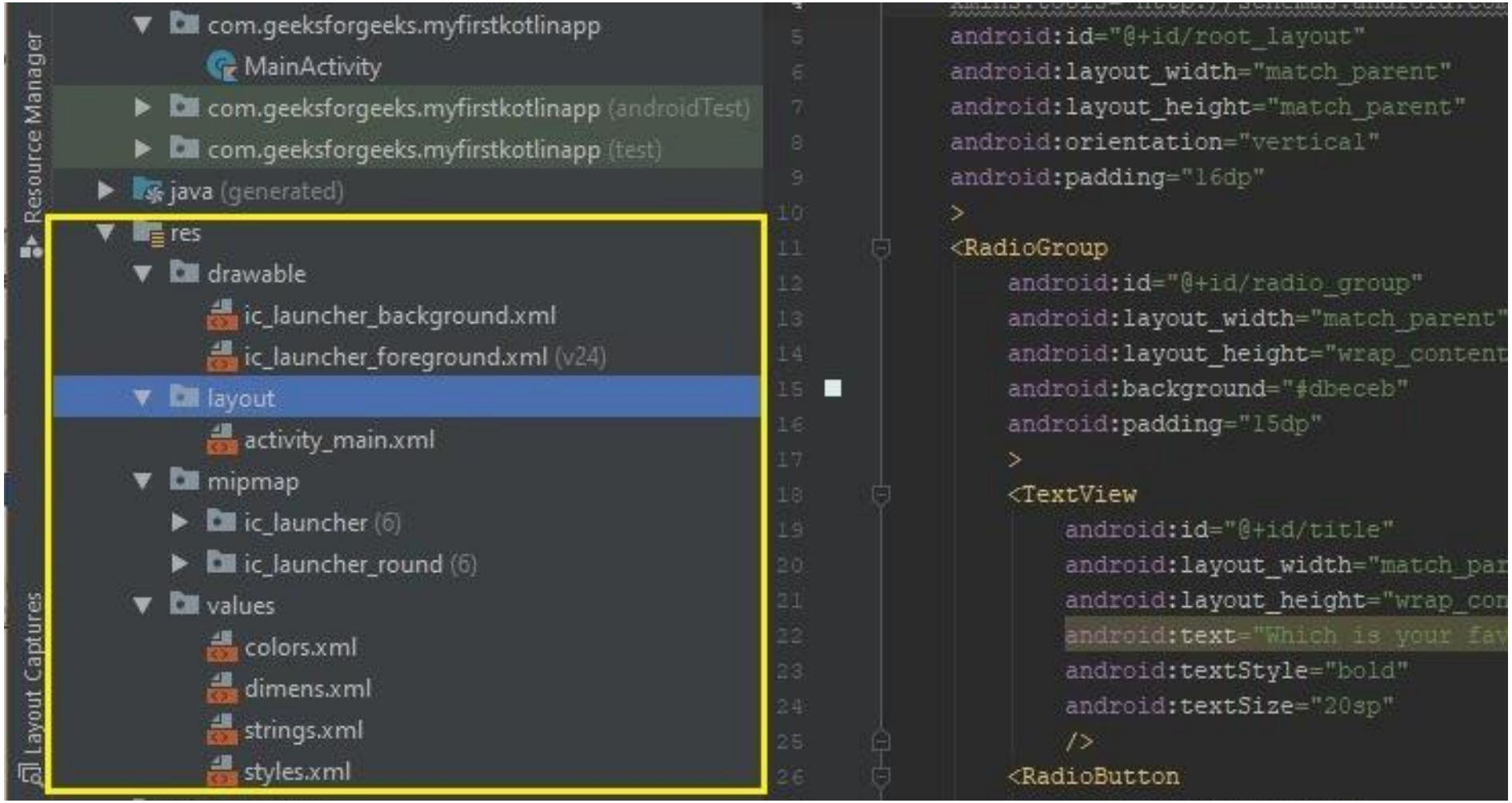
Java folder

Java folder contains all the java and Kotlin source code (.java) files that we create during the app development, including other Test files. If we create any new project using Kotlin, by default the class file MainActivity.kt file will create automatically under the package name “com.geeksforgeeks.myfirstkotlinapp” like as shown below.



Resource (res) folder:

The resource folder is the most important folder because it contains all the non-code sources like images, XML layouts, UI strings for our android application.



res/drawable folder

It contains the different types of images used for the development of the application. We need to add all the images in a drawable folder for the application development.

res/layout folder

The layout folder contains all XML layout files which we used to define the user interface of our application. It contains the **activity_main.xml** file.

res/midmap folder

This folder contains launcher.xml files to define icons that are used to show on the home screen. It contains different density types of icons depending upon the size of the device such as hdpi, mdpi, xhdpi.

res/values folder

Values folder contains a number of XML files like strings, dimensions, colors and styles definitions. One of the most important file is **strings.xml** file which contains the resources.

Gradle Scripts folder

Gradle means automated build system and it contains number of files which are used to define a build configuration which can be apply to all modules in our application. In build.gradle (Project) there are buildscripts and in build.gradle (Module) plugins and implementations are used to build configurations that can be applied to all our application modules.

The Application Manifest File | Android

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.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an xmls:android attribute that will supply several system attributes used within the file. We use the versionCode attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the versionName attribute is used to specify a public version that will be displayed to the users.

We can also specify whether our app should install on an SD card of the internal memory using the installLocation attribute. A typical manifest node looks as:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paad.myapp"
    android:versionCode="1"
    android:versionName="0.9 Beta"
    android:installLocation="preferExternal">
    [....manifest nodes....]
</manifest>
```

1. uses-sdk

It is used to define a minimum and maximum SDK version that must be available on a device so that our application functions properly, and target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively

```
<uses-sdk android:minSdkVersion=”6”
          android:targetSdkVersion=”15”/>
```

2. uses-configuration

The uses-configuration nodes are used to specify the combination of input mechanisms that are supported by our application. It is useful for games that require particular input controls.

```
<uses-configuration android:reqTouchScreen=”finger”
        android:reqNavigation=”trackball”
        android:reqHardKeyboard=”true”
        android:reqKeyboardType=”qwerty”/>
<uses-configuration android:reqTouchScreen=”finger”
        android:reqNavigation=”trackball”
        android:reqHardKeyboard=”true”
      android:reqKeyboardType=”twelvekey”/>
```

3. uses-features

It is used to specify which hardware features your application requirement. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware, as follows:

<**uses-feature** android:name=”android.hardware.nfc”/>

4. supports-screens

It is used to describe the screen support for our application:

```
<supports-screens
  android:smallScreens=”false”
  android:normalScreens=”true”
  android:largeScreens=”true”
  android:xlargeScreens=”true”/>
```

5. permission

It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

```
<permission
  android: name=”com.paad.DETONATE_DEVICE”
  android:protectionLevel=“dangerous”
  android:label=”Self Destruct”
  android:description=”@string/detonate_description”>
</permission>
```

Android | res/values folder

The **res/values folder** is used to store the values for the resources that are used in many Android projects to include features of color, styles, dimensions etc.
Below explained are few basic files, contained in the res/values folder:
1.colors.xml: The *colors.xml* is an XML file which is used to store the colors for the resources.An Android project contains 3 essential colours namely:

1.colorPrimary
2.colorPrimaryDark
3.colorAccent

2.These colors are used in some predefined resources of the android studio as well. These colors as needed to be set opaque otherwise it could result in some exceptions to arise.
3.Below mentioned is the implementation of colors.xml resource:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#1294c8</color>
    <color name="colorPrimaryDark">#1294c8</color>
    <color name="colorAccent">#FF4081</color>

    <color name="text_color">#555555</color>


    <color name="colorText">#FFFFFF</color>
    <color name="colorTextHint">#51d8c7</color>
</resources>
```

dimens.xml: The *dimens.xml* is used for defining the dimensions for different widgets to be included in the Android project. It is a good coding practice to use dimens.xml to define a dimension rather than just writing the dimension in the resource, due to the fact that if ever any need arises to change the dimension, instead of making a change to all, only the dimens.xml can be changed once and the change is reflected in all.
Below mentioned is the implementation of dimens.xml resource:

```
<resources>
    <!-- Default screen margins, per the Android Design
guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="nav_header_vertical_spacing">8dp</dimen>
    <dimen name="nav_header_height">176dp</dimen>
    <dimen name="fab_margin">16dp</dimen>
</resources>
```

strings.xml: One of the most important as well as widely used values file is the strings.xml due to its applicability in the Android project. Basic function of the strings.xml is to define the strings in one file so that it is easy to use same string in different positions in the android project plus it makes the project looks less messy. We can also define an array in this file as well.
Below mentioned is the implementation of strings.xml resource:

```
<resources>
    <string name="app_name">Workshop app</string>

    <string name="navigation_drawer_open">Open navigation drawer</string>
    <string name="navigation_drawer_close">Close navigation drawer</string>
    <string name="action_settings">Settings</string>
    <string name="hello_blank_fragment">Hello blank fragment</string>
    <string name="date">Date:</string>
    <string name="timings">Timings:</string>
</resources>
```

styles.xml: Another important file in the values folder is the *styles.xml* where all the themes of the Android project are defined. The base theme is given by default having the option to customize or make changes to the customized theme as well. Every theme has a parent attribute which defines the base of the theme. There are a lot of options to choose from depending on the need of the Android project.
Below mentioned is the implementation of styles.xml resource:

```
<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>
    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>
    <style name="AppTheme.AppBarOverlay"
parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />
</resources>
```

Components of an Android Application

There are some necessary building blocks that an Android application consists of. These loosely coupled components are bound by the application manifest file which contains the description of each component and how they interact. The manifest file also contains the app's metadata, its hardware configuration, and platform requirements, external libraries, and required permissions. There are the following main components of an android app:

1. Activities

Activities are said to be the presentation layer of our applications. The UI of our application is built around one or more extensions of the Activity class. By using Fragments and Views, activities set the layout and display the output and also respond to the user's actions. An activity is implemented as a subclass of class Activity

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

Services

Services are like invisible workers of our app. These components run at the backend, updating your data sources and Activities, triggering Notification, and also broadcast Intents. They also perform some tasks when applications are not active. A service can be used as a subclass of class Service:

```
public class ServiceName extends Service {
}
```

Content Providers

It is used to manage and persist the application data also typically interacts with the SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured. A content provider should be a sub-class of the class ContentProvider.

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

Broadcast Receivers

They are known to be intent listeners as they enable your application to listen to the Intents that satisfy the matching criteria specified by us. Broadcast Receivers make our application react to any received Intent thereby making them perfect for creating event-driven applications.

Intents

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

Widgets

These are the small visual application components that you can find on the home screen of the devices. They are a special variation of Broadcast Receivers that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

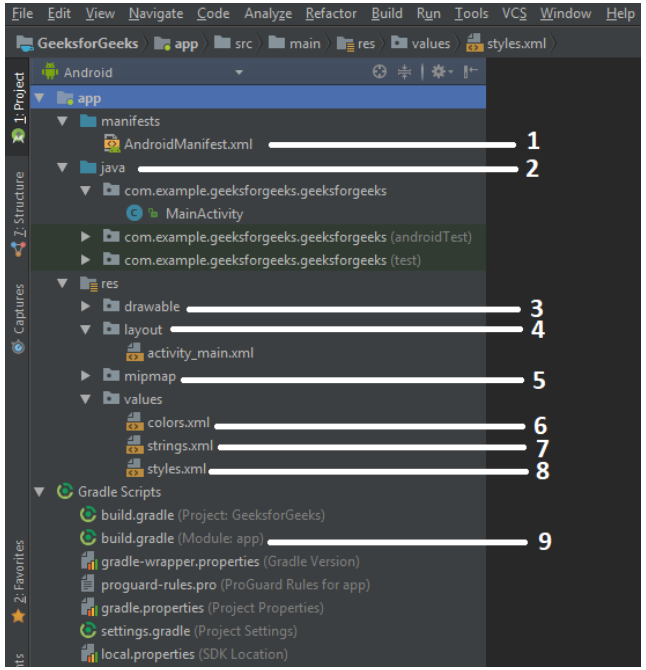
Notifications

Notifications are the application alerts that are used to draw the user's attention to some particular app event without stealing focus or interrupting the current activity of the user. They are generally used to grab user's attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: E-mail popups, Messenger popups, etc

Android | Android Application File Structure

It is very important to know about the basics of Android Studio's file structure. In this article, some important files/folders, and their significance is explained for the easy understanding of the Android studio work environment.

In the below image, several important files are marked:



All of the files marked in the above image are explained below in brief:

1. 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.
A typical AndroidManifest.xml file looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/androi
d"
package="com.example.geeksforgeeks.geeksforgeeks">
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

2. 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.

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

4.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.

Below is a sample layout file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

5. 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.

6. 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>
```

7. 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">GeeksforGeeks</string>
</resources>
```

8. 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>
```

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

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.geeksforgeeks.geeksforgeeks"
        minSdkVersion 16
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:0.5'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:2.2.2'
}
```

Android System Architecture

The Android software stack generally consists of a Linux kernel and a collection of C/C++ libraries that is exposed through an application framework that provides services, and management of the applications and run time.

Linux Kernel

Android was created on the open source kernel of Linux. One main reason for choosing this kernel was that it provided proven core features on which to develop the Android operating system. The features of Linux kernel are:

1. Security:

The Linux kernel handles the security between the application and the system.

2.Memory Management:

Click to add text

It efficiently handles the memory management thereby providing the freedom to develop our apps.

3.Process Management:

It manages the process well, allocates resources to processes whenever they need them.

4.Network Stack:

It effectively handles the network communication.

5.Driver Model:

It ensures that the application works. Hardware manufacturers can build their drivers into the Linux build.

Libraries:

Running on the top of the kernel, the Android framework was developed with various features. It consists of various C/C++ core libraries with numerous of open source tools. Some of these are:

1.The Android runtime:

The Android runtime consist of core libraries of Java and ART(the Android RunTime). Older versions of Android (4.x and earlier) had Dalvik runtime.

2.Open GL(graphics library):

This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.

3.WebKit:

This open source web browser engine provides all the functionality to display web content and to simplify page loading.

4.Media frameworks:

These libraries allow you to play and record audio and video.

5.Secure Socket Layer (SSL):

These libraries are there for Internet security.

Android Runtime:

It is the third section of the architecture. It provides one of the key components which is called Dalvik Virtual Machine. It acts like Java Virtual Machine which is designed specially for Android. Android uses it's own custom VM designed to ensure that multiple instances run efficiently on a single device.

The Delvik VM uses the device's underlying Linux kernel to handle low-level functionality,including security, threading and memory management.

Application Framework

The Android team has built on a known set proven libraries, built in the background, and all of it these is exposed through Android interfaces. These interfaces warp up all the various libraries and make them useful for the Developer. They don't have to build any of the functionality provided by the android. Some of these interfaces include:

1.Activity Manager:

It manages the activity lifecycle and the activity stack.

2.Telephony Manager:

It provides access to telephony services as related subscriber information, such as phone numbers.

3.View System:

It builds the user interface by handling the views and layouts.

4.Location manager:

It finds the device's geographic location.

Applications:

Android applications can be found at the topmost layer. At application layer we write our application to be installed on this layer only. Examples of applications are Games, Messages, Contacts etc.