ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

VI – SEMESTER

08EP609 : Mobile App Development Lab

Name : ____

Reg. No. : ____

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

VI – SEMESTER

08EP609 : Mobile App Development Lab

Certified that this is a bona fide record of work done by Mr./Ms.____

Reg. No.____

of B.E. (Computer Science and Engineering) in the 08EP609: Mobile App Development Laboratory during the even semester of the academic year 2018 – 2019.

Staff in-charge

Internal Examiner External Examiner

Annamalainagar
Date: … /…. / 2019.

CONTENTS

Ex. No. 01.
Date:

Study of Android Studio IDE

Introduction:

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 the productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where developer can develop for all Android devices
- Instant Run to push changes to the running app without building a new APK
- Code templates and GitHub integration to help the developer to 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 Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

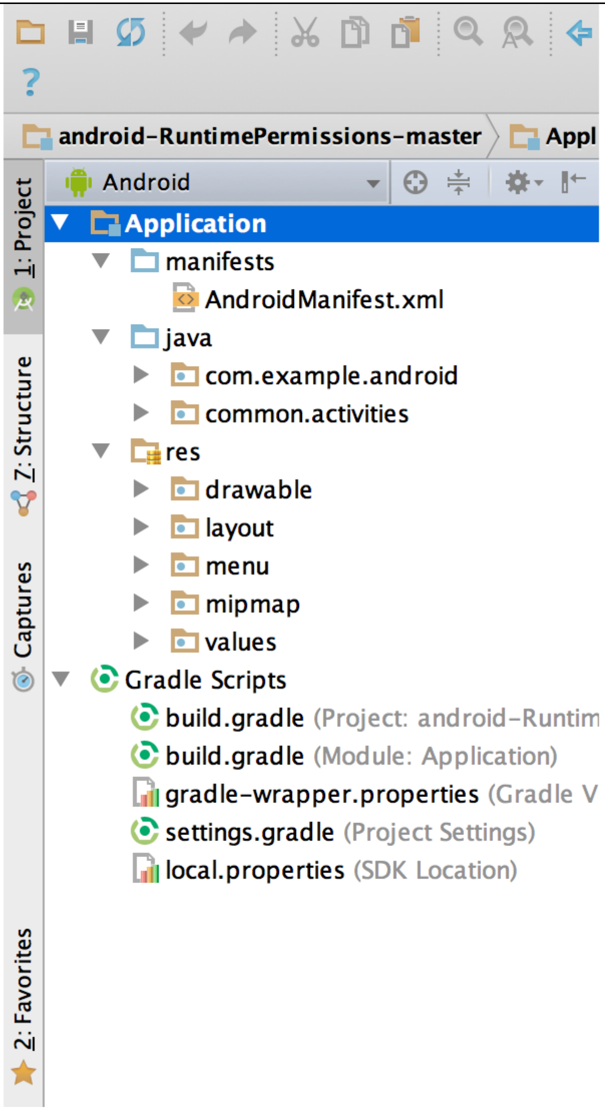
- Android app modules
- Library modules
- Google App Engine modules
-

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

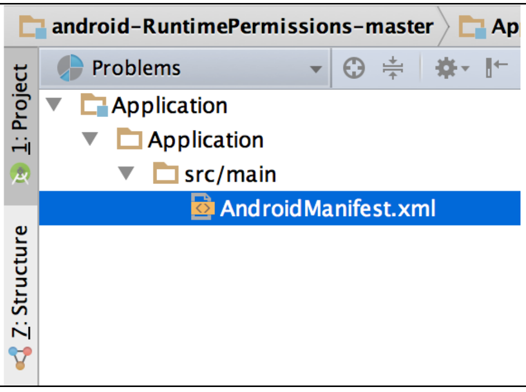
The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project dropdown (showing as Android).



```
▼ Application
    ▼ manifests
        AndroidManifest.xml
    ▼ java
        ▶ com.example.android
        ▶ common.activities
    ▼ res
        ▶ drawable
        ▶ layout
        ▶ menu
        ▶ mipmap
        ▶ values
▼ Gradle Scripts
    build.gradle (Project: android-Runtim
    build.gradle (Module: Application)
    gradle-wrapper.properties (Gradle V
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

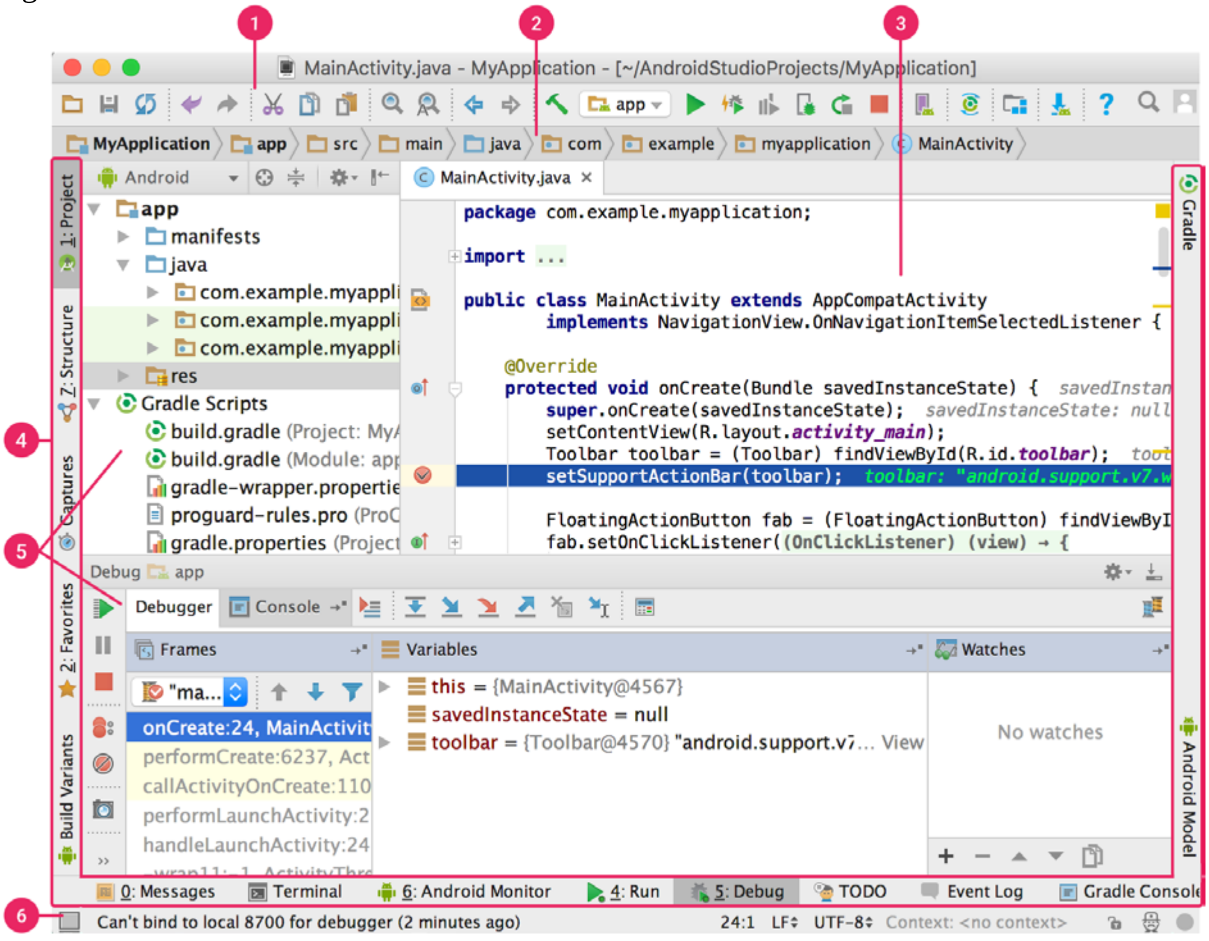
Developer can also customize the view of the project files to focus on specific aspects of the app development. For example, selecting the Problems view of any project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a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

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



The user interface

The Android Studio main window is made up of several logical areas identified in figure. The Android Studio main window.



```
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {  savedInstan
        super.onCreate(savedInstanceState);  savedInstanceState: null
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  tool
        setSupportActionBar(toolbar);  toolbar: "android.support.v7.w

        FloatingActionButton fab = (FloatingActionButton) findViewByI
        fab.setOnClickListener((OnClickListener) (view) → {
```

```
this = {MainActivity@4567}
savedInstanceState = null
toolbar = {Toolbar@4570} "android.support.v7... View
```

```
Can't bind to local 8700 for debugger (2 minutes ago)     24:1  LF÷  UTF-8÷  Context: <no context>
```

1. The **toolbar** used to carry out a wide range of actions, including running an app and launching Android tools.
2. The **navigation bar** helps to navigate through the 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 developer can 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 allows the developer to expand or collapse individual tool windows.

5. The **tool windows** give the access to specific tasks like project management, search, version control, and more. Developer can expand them and collapse them.
6. The **status bar** displays the status of the project and the IDE itself, as well as any warnings or messages.

Developer can organize the main window to give him more screen space by hiding or moving toolbars and tool windows. Developer can also use keyboard shortcuts to access most IDE features.

At any time, one can search across the source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful, when the developer trying to locate a particular IDE action.

Tool windows

Instead of using pre-set perspectives, Android Studio follows the developer context and automatically brings up relevant tool windows as he 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. Developer 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 the 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.

Developer can also use keyboard shortcuts to open tool windows. Following Table lists the shortcuts for the most common 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 the developer wants 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**.

Developer 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 the search query.

Code completion

Android Studio has three types of code completion, which can be accessed using keyboard shortcuts.

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+Space**	**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+Space**
Statement Completion	Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.	**Control+Shift+Enter**	**Shift+Command+Enter**

Developer can also perform quick fixes and show intention actions by pressing **Alt+Enter**.

Sample code

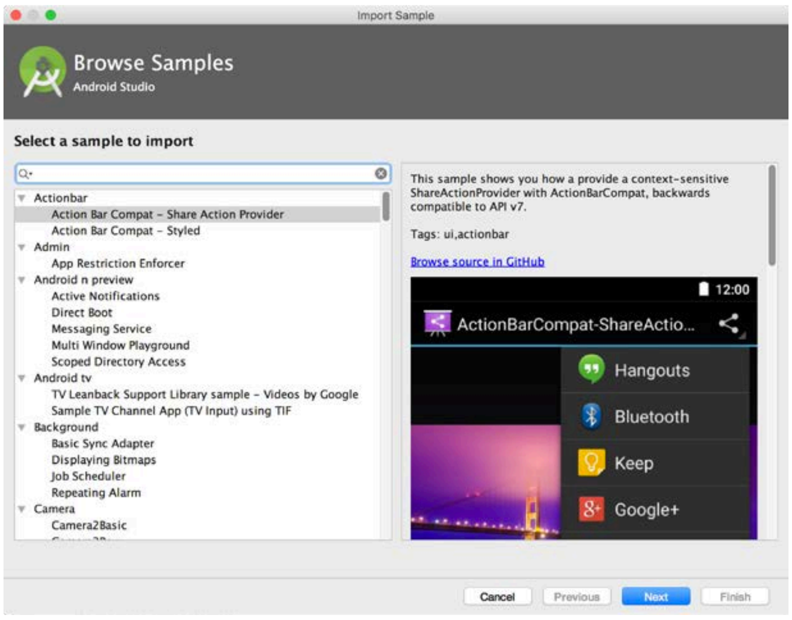
The Code Sample Browser in Android Studio helps to find high-quality, Google-provided Android code samples based on the currently highlighted symbol in the project. Android Studio provides a selection of code samples and templates to accelerate the app development. Browse sample code to learn how to build different components for the applications. Use templates to create new app modules, individual activities, or other specific Android project components.

In the Browse Samples dialog

Developer can use the samples browser to select, preview, and import one or more sample apps as projects. Developer can also browse the source code through GitHub.

1. Select **File > New > Import Sample**.
2. Use the search box or the scroll bar to browse the samples.
3. When you find a sample that interests you, highlight it and take a look at the preview.
4. If you want to import it as a project, click **Next** and then **Finish**.

Browse Samples dialog with sample highlighted in the left column and previewed in the right column.



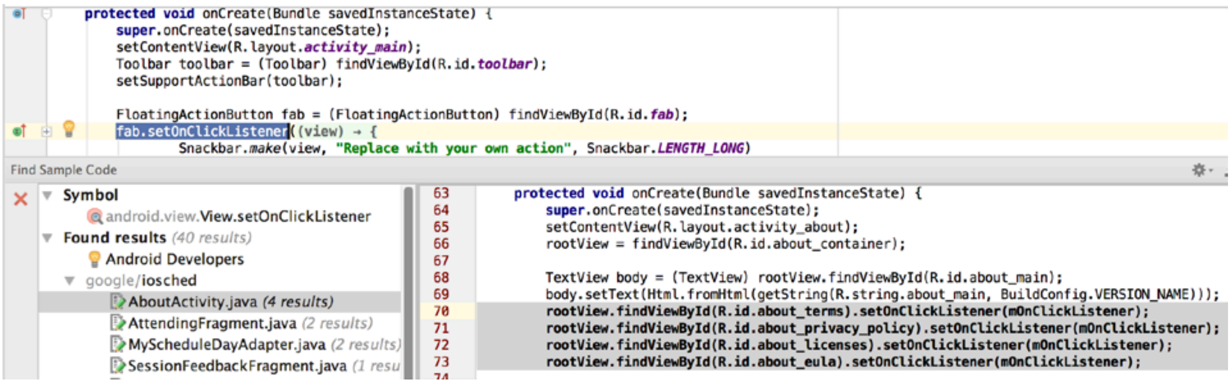
Inline from the editor

The Code Sample Browser in Android Studio helps to find Android code samples based on the currently highlighted symbol in the project.

1. In the code, highlight a variable, type, or method.
2. Right-click to display the context menu.
3. From the context menu, select **Find Sample Code**. (The results of the search appear in a tool window as shown.)
4. In the left pane of the tool window, select a sample.

Then, scroll through the right pane to find highlighted code lines that are used in the selected sample.

The Code Sample Browser.



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener((view) → {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
```

Find Sample Code
Symbol
android.view.View.setOnClickListener
Found results (40 results)
Android Developers
google/iosched
AboutActivity.java (4 results)
AttendingFragment.java (2 results)
MyScheduleDayAdapter.java (2 results)
SessionFeedbackFragment.java (1 resu

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_about);
    rootView = findViewById(R.id.about_container);

    TextView body = (TextView) rootView.findViewById(R.id.about_main);
    body.setText(Html.fromHtml(getString(R.string.about_main, BuildConfig.VERSION_NAME)));
    rootView.findViewById(R.id.about_terms).setOnClickListener(mOnClickListener);
    rootView.findViewById(R.id.about_privacy_policy).setOnClickListener(mOnClickListener);
    rootView.findViewById(R.id.about_licenses).setOnClickListener(mOnClickListener);
    rootView.findViewById(R.id.about_eula).setOnClickListener(mOnClickListener);
```

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** 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.

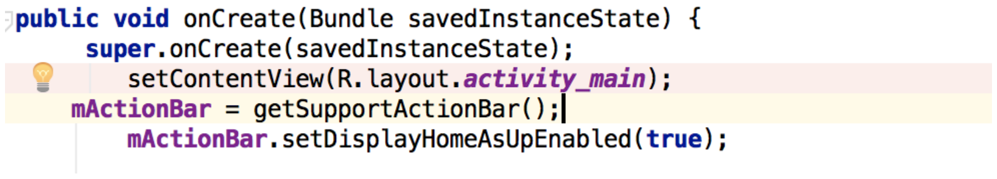
- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12.** Using this action, you can quickly navigate to any part of your current file.
- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N**. 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**. 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**.
- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7**.

Style and formatting

During editing, Android Studio automatically applies formatting and styles as specified in the code style settings. One 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 the code style settings, click **File** > **Settings** > **Editor** > **Code Style**.

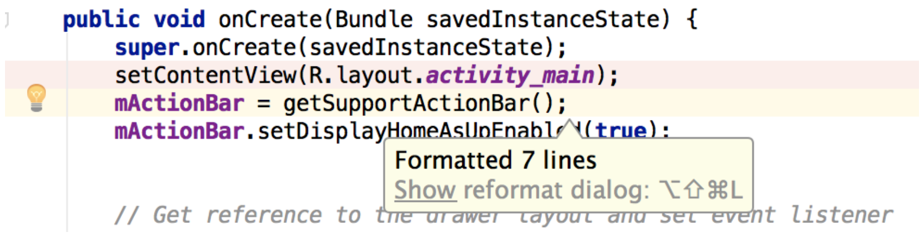
Although the IDE automatically applies formatting, one can also explicitly call the *Reformat Code* action by pressing **Control+Alt+L**, or auto-indent all lines by pressing **Control+Alt+I**.

Code before formatting.



```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
  mActionBar = getSupportActionBar();
        mActionBar.setDisplayHomeAsUpEnabled(true);
```

Code after formatting.



```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mActionBar = getSupportActionBar();
    mActionBar.setDisplayHomeAsUpEnabled(true);


    // Get reference to the drawer layout and set event listener
```

Formatted 7 lines
Show reformat dialog: ⌥⇧⌘L

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 the 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:

1. From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
2. 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 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. Developer 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 source sets.

By employing the flexibility of Gradle, one can achieve all of this without modifying the 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 importing an existing project, Android Studio automatically generates the necessary build files.

Build variants

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

Multiple APK support

Multiple APK support allows to efficiently create multiple APKs based on screen density or ABI. For example, one 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.

Resource shrinking

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

Managing dependencies

Dependencies for the project are specified by name in the build.gradle file. Gradle takes care of finding the dependencies and making them available in the build. Developer can declare module dependencies, remote binary dependencies, and local binary dependencies in the 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.

Debug and profile tools

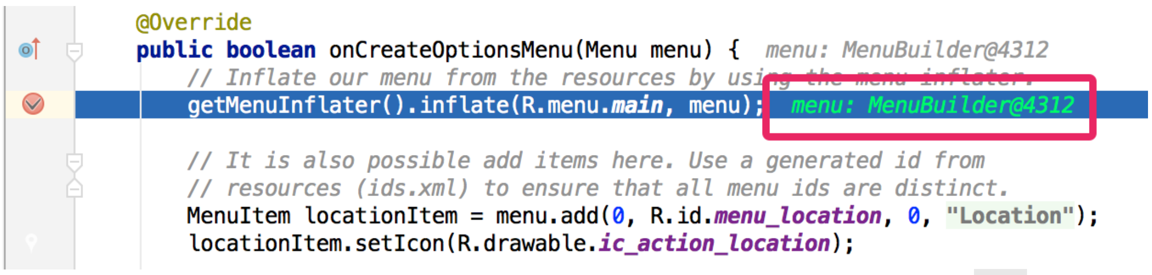
Android Studio assists the developer in debugging and improving the performance of the code, including inline debugging and performance analysis tools.

Inline debugging

Use of inline debugging is to enhance the 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

An inline variable value is shown below.



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {  menu: MenuBuilder@4312
    // Inflate our menu from the resources by using the menu inflater.
    getMenuInflater().inflate(R.menu.main, menu);  menu: MenuBuilder@4312

    // It is also possible add items here. Use a generated id from
    // resources (ids.xml) to ensure that all menu ids are distinct.
    MenuItem locationItem = menu.add(0, R.id.menu_location, 0, "Location");
    locationItem.setIcon(R.drawable.ic_action_location);
```

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 to more easily track the app's memory and CPU usage, find de-allocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With the app running on a device or emulator, open the **Android Profiler** tab.

Heap dump

When profiling memory usage in Android Studio, developer 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

Develope can use Memory Profiler to track memory allocation and watch where objects are being allocated when he perform certain actions. Knowing these allocations enables him to optimize his app's performance and memory use by adjusting the method calls related to those actions.

Data file access

The Android SDK tools, such as Systrace, and logcat, generate performance and debugging data for detailed app analysis.

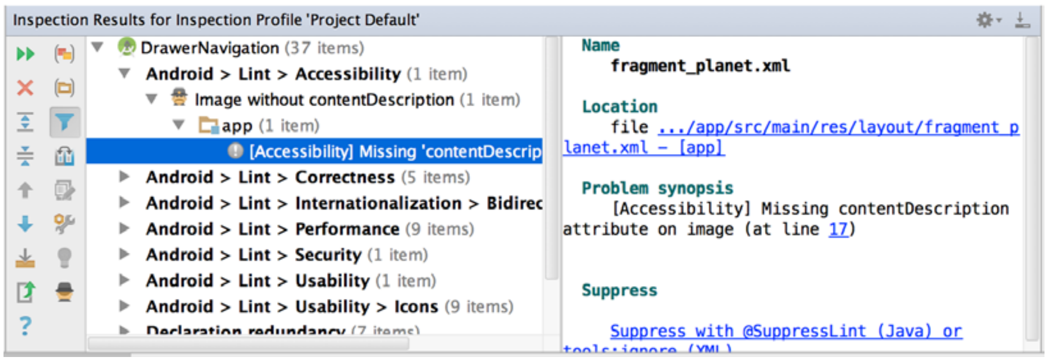
To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to the standard investigate your RAM usage file format.

Code inspections

Whenever compiling the program, Android Studio automatically runs configured Lint and other IDE inspections to help the developer to easily identify and correct problems with the structural quality of the code.

The Lint tool checks the Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

The results of a Lint inspection in Android Studio.



```
Inspection Results for Inspection Profile 'Project Default'

▼ DrawerNavigation (37 items)
  ▼ Android > Lint > Accessibility (1 item)
    ▼ Image without contentDescription (1 item)
      ▼ app (1 item)
          [Accessibility] Missing 'contentDescrip
  ▶ Android > Lint > Correctness (5 items)
  ▶ Android > Lint > Internationalization > Bidirec
  ▶ Android > Lint > Performance (9 items)
  ▶ Android > Lint > Security (1 item)
  ▶ Android > Lint > Usability (1 item)
  ▶ Android > Lint > Usability > Icons (9 items)
  ▶ Declaration redundancy (7 items)

Name
    fragment_planet.xml

Location
    file .../app/src/main/res/layout/fragment_p
lanet.xml - [app]

Problem synopsis
    [Accessibility] Missing contentDescription
attribute on image (at line 17)


Suppress

    Suppress with @SuppressLint (Java) or
tools:ignore (XML)
```

In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline the coding workflow.

Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help in catching 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.

Log messages

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

Performance profiling

Developer can profile the app's CPU, memory, and network performance, by opening the Android Profiler, by clicking **View** > **Tool Windows** > **Android Profiler**.

Installing Android Studio

Setting up Android Studio takes just a few clicks.

Windows

To install Android Studio on Windows, proceed as follows:

1. If downloaded an .exe file (recommended), double-click to launch it.
   If downloaded a .zip file, unpack the ZIP, copy the **android-studio** folder into the **Program Files** folder, and then open the **android-studio** > **bin** folder and launch studio64.exe (for 64-bit machines) or studio.exe (for 32-bit machines).
2. Follow the setup wizard in Android Studio and install any SDK packages that it recommends.

Configure Android Studio

Android Studio provides wizards and templates that verify the system requirements, such as the Java Development Kit (JDK) and available RAM, and configure default settings, such as optimized default Android Virtual Device (AVD) emulation and updated system images.
Android Studio provides access to two configuration files through the **Help** menu:

- studio.vmoptions: Customize options for Studio's Java Virtual Machine (JVM), such as heap size and cache size.

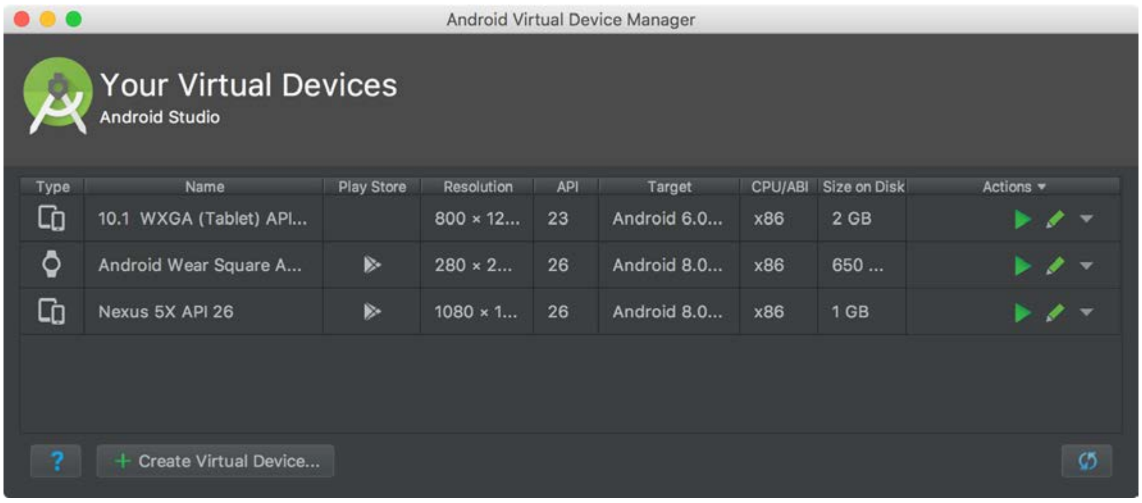
- idea.properties: Customize Android Studio properties, such as the plugins folder path or maximum supported file size.

Create and manage virtual devices

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, or Android TV device that developer wants to simulate in the Android Emulator. The AVD Manager is an interface can be launched from Android Studio that helps to create and manage AVDs.

To open the AVD Manager, do one of the following:

- Select **Tools > AVD Manager**.
- Click **AVD Manager** in the toolbar.


Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	10.1 WXGA (Tablet) API...		800 × 12...	23	Android 6.0...	x86	2 GB	
	Android Wear Square A...	▶	280 × 2...	26	Android 8.0...	x86	650 ...	
	Nexus 5X API 26	▶	1080 × 1...	26	Android 8.0...	x86	1 GB	

About AVDs

An AVD contains a hardware profile, system image, storage area, skin, and other properties. It is recommended to create an AVD for each system image that an app could potentially support based on the <uses-sdk> setting in the manifest.

Hardware profile

The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Pixel devices, and can define or customize the hardware profiles as needed.

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

System images

A system image labeled with **Google APIs** includes access to Google Play services. A system image labeled with the Google Play logo in the **Play Store** column includes the Google Play Store app *and* access to Google Play services, including a **Google Play** tab in the **Extended controls** dialog that provides a convenient button for updating Google Play services on the device.

To ensure app security and a consistent experience with physical devices, system images with the Google Play Store included are signed with a release key, which means that you cannot get elevated privileges (root) with these images. If developer require elevated privileges (root) to aid with the app troubleshooting, use the Android Open Source Project (AOSP) system images that do not include Google apps or services.

Storage area

The AVD has a dedicated storage area on the development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card. If needed, use the AVD Manager to wipe user data, so the device has the same data as if it were new.

Skin

An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. Developer can also define his own or use skins provided by third parties.

AVD and app features

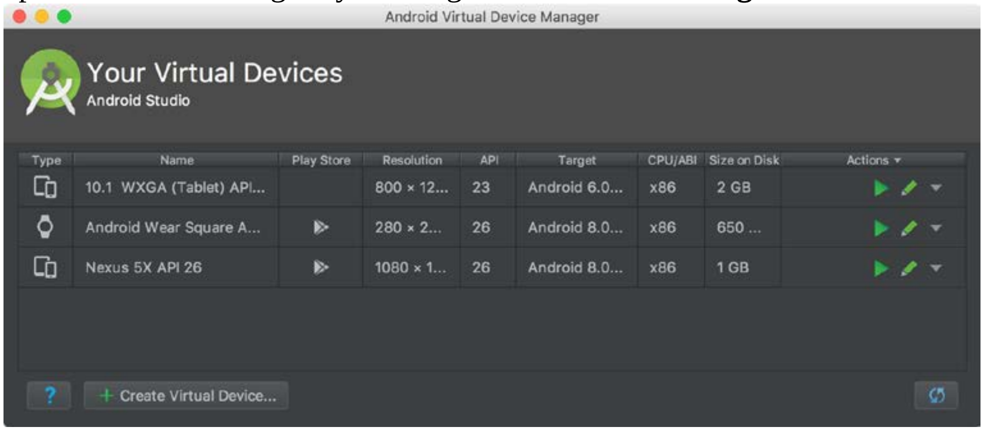
Be sure the AVD definition includes the device features of the app depend on. Hardware Profile Properties and AVD Properties for lists of features can be defined in the AVDs.

Create an AVD

To launch an app into an emulator, instead run the app from Android Studio and then in the **Select Deployment Target** dialog that appears, click **Create New Virtual Device**.

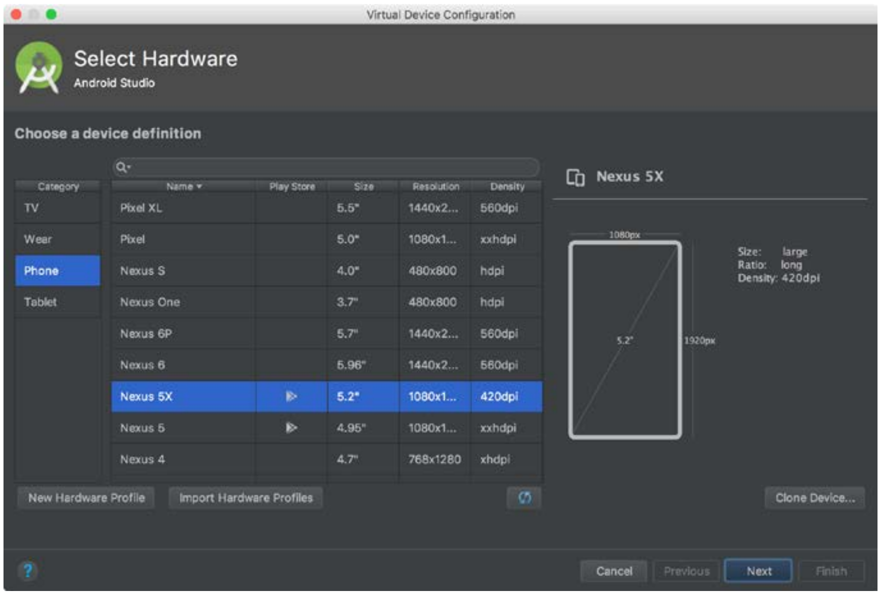
To create a new AVD:

1. Open the AVD Manager by clicking **Tools > AVD Manager**.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	10.1 WXGA (Tablet) API...		800 × 12...	23	Android 6.0...	x86	2 GB	
	Android Wear Square A...	▶	280 × 2...	26	Android 8.0...	x86	650 ...	
	Nexus 5X API 26	▶	1080 × 1...	26	Android 8.0...	x86	1 GB	

2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog.
   The **Select Hardware** page appears.
   Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.



Choose a device definition

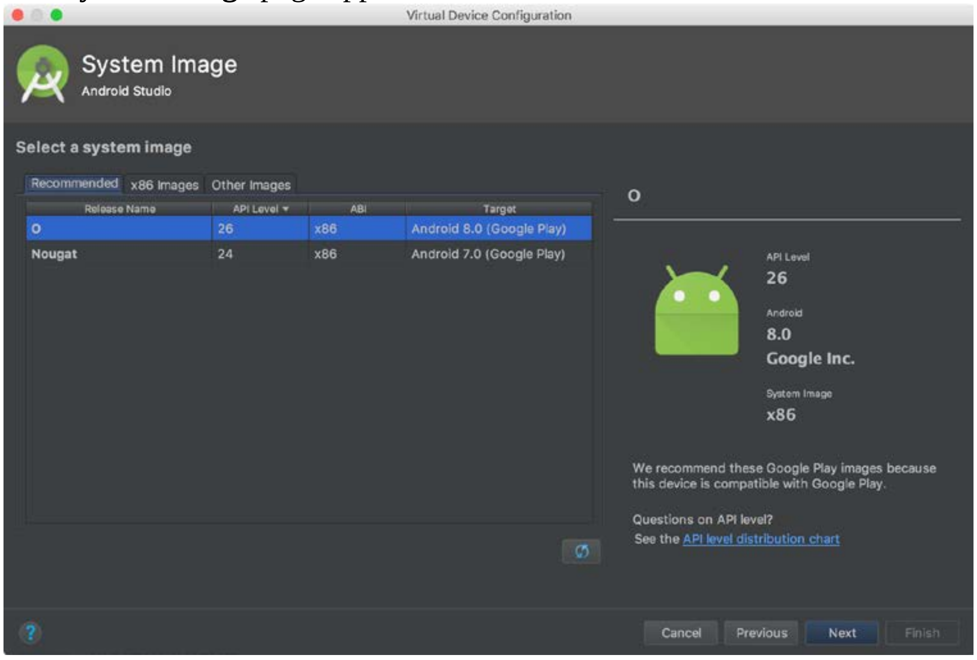
Category
TV
Wear
Phone
Tablet

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2...	560dpi
Pixel		5.0"	1080x1...	xxhdpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2...	560dpi
Nexus 6		5.96"	1440x2...	560dpi
Nexus 5X	▶	5.2"	1080x1...	420dpi
Nexus 5	▶	4.95"	1080x1...	xxhdpi
Nexus 4		4.7"	768x1280	xhdpi

Nexus 5X

Size: large
Ratio: long
Density: 420dpi

3. Select a hardware profile, and then click **Next**.
   If you don't see the hardware profile you want, you can create or import a hardware profile.
   The **System Image** page appears.



Select a system image

Release Name	API Level	ABI	Target
O	26	x86	Android 8.0 (Google Play)
Nougat	24	x86	Android 7.0 (Google Play)

O

API Level
26
Android
8.0
Google Inc.
System Image
x86

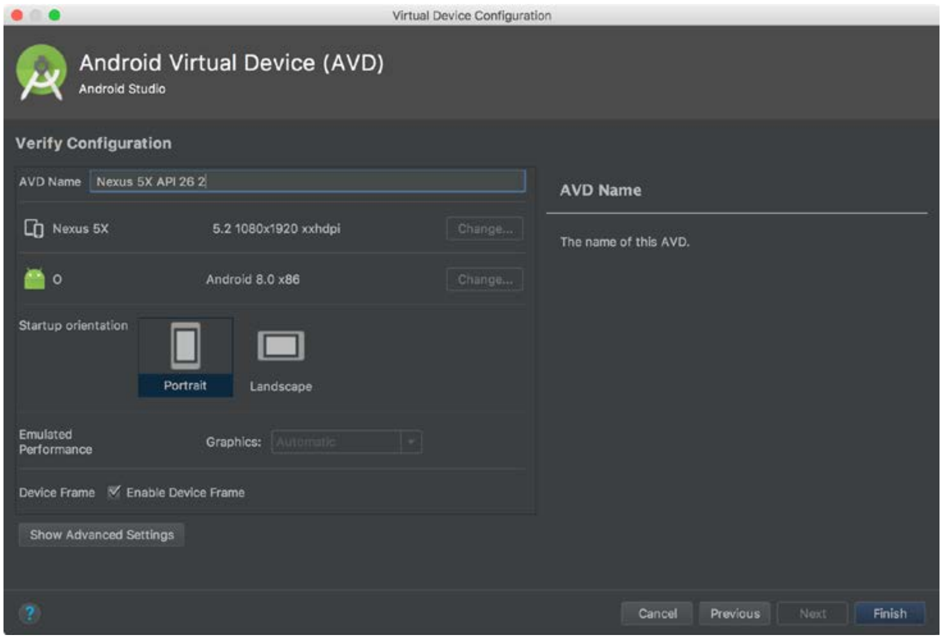
We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

4. Select the system image for a particular API level, and then click **Next**.
   The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.
   If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.
   The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the minSdkVersion attribute of the app manifest file. For more information about the relationship between system API level and minSdkVersion, see Versioning Your Apps.
   If your app declares a <uses-library> element in the manifest file, the app requires a system image in which that external library is present. If you want to run

your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The **Verify Configuration** page appears.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Nexus 5X API 26 2

Nexus 5X — 5.2 1080x1920 xxhdpi — Change...

O — Android 8.0 x86 — Change...

Startup orientation: Portrait / Landscape

Emulated Performance — Graphics: Automatic

Device Frame: ✓ Enable Device Frame

Show Advanced Settings

AVD Name

The name of this AVD.

5. Change AVD properties as needed, and then click **Finish**.
   Click **Show Advanced Settings** to show more settings, such as the skin.
   The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.

To create an AVD starting with a copy:

1. From the **Your Virtual Devices** page of the AVD Manager, right-click an AVD and select **Duplicate**.
   Or click Menu ▼ and select **Duplicate**.
   The **Verify Configuration** page appears.
2. Click **Change** or **Previous** if you need to make changes on the **System Image** and **Select Hardware** pages.
3. Make your changes, and then click **Finish**.
   The AVD appears in the **Your Virtual Devices** page.

Create a hardware profile

The AVD Manager provides predefined hardware profiles for common devices so you can easily add them to your AVD definitions. If you need to define a different device, you can create a new hardware profile. You can define a new hardware profile from the beginning, or copy a hardware profile as a start. The preloaded hardware profiles aren't editable.

To create a new hardware profile from the beginning:

1. In the **Select Hardware** page, click **New Hardware Profile**.
2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click **Finish**.

Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

To create a hardware profile starting with a copy:

1. In the **Select Hardware** page, select a hardware profile and click **Clone Device**.
   Or right-click a hardware profile and select **Clone**.
2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click **Finish**.
   Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

Edit existing AVDs

From the **Your Virtual Devices** page, you can perform the following operations on an existing AVD:

- To edit an AVD, click **Edit this AVD** ✎ and make your changes.
- To delete an AVD, right-click an AVD and select **Delete**. Or click Menu ▼ and select **Delete**.
- To show the associated AVD .ini and .img files on disk, right-click an AVD and select **Show on Disk**. Or click Menu ▼ and select **Show on Disk**.
- To view AVD configuration details that you can include in any bug reports to the Android Studio team, right-click an AVD and select **View Details**. Or click Menu ▼ and select **View Details**.

Edit existing hardware profiles

From the **Select Hardware** page, you can perform the following operations on an existing hardware profile:

- To edit a hardware profile, select it and click **Edit Device**. Or right-click a hardware profile and select **Edit**. Next, make your changes.
- To delete a hardware profile, right-click it and select **Delete**.

You can't edit or delete the predefined hardware profiles.

Run and stop an emulator, and clear data

From the **Your Virtual Devices** page, you can perform the following operations on an emulator:

- To run an emulator that uses an AVD, double-click the AVD. Or click **Launch** ▶.
- To stop a running emulator, right-click an AVD and select **Stop**. Or click Menu ▼ and select **Stop**.
- To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select **Wipe Data**. Or click Menu ▼ and select **Wipe Data**.

Import and export hardware profiles

From the **Select Hardware** page, you can import and export hardware profiles:

- To import a hardware profile, click **Import Hardware Profiles** and select the XML file containing the definition on your computer.
- To export a hardware profile, right-click it and select **Export**. Specify the location where you want to store the XML file containing the definition.

Hardware profile properties

You can specify the following properties of hardware profiles in the **Configure Hardware Profile** page. AVD configuration properties override hardware profile properties, and emulator properties that you set while the emulator is running override them both.
The predefined hardware profiles included with the AVD Manager aren't editable. However, you can copy them and edit the copies.

Hardware Profile Property	**Description**
Device Name	Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name.
Device Type	Select one of the following: • Phone/Tablet • Wear OS • Android TV
Screen Size	The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it's reduced in size at launch.
Screen Resolution	Type a width and height in pixels to specify the total number of pixels on the simulated screen.
Round	Select this option if the device has a round screen, such as some Wear OS devices.
Memory: RAM	Type a RAM size for the device and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Input: Has Hardware Buttons (Back/Home/Menu)	Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won't appear on the screen. You can use the emulator side panel to "press" the buttons, in either case.
Input: Has Hardware Keyboard	Select this option if your device has a hardware keyboard. Deselect it if it doesn't. If you select this option, a keyboard won't appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case.
Navigation Style	Select one of the following: • None - No hardware controls. Navigation is through the software. • D-pad - Directional Pad support. • Trackball • Wheel These options are for actual hardware controls on the device itself. However, the events sent to the device by an external controller are the same.
Supported Device States	Select one or both options: • Portrait - Oriented taller than wide. • Landscape - Oriented wider than tall. If you select both, you can switch between orientations in the

	emulator. You must select at least one option to continue.
Cameras	To enable the camera, select one or both options: • Back-Facing Camera - The lens faces away from the user. • Front-Facing Camera - The lens faces toward the user. Later, you can use a webcam or a photo provided by the emulator to simulate taking a photo with the camera.
Sensors: Accelerometer	Select if the device has hardware that helps the device determine its orientation.
Sensors: Gyroscope	Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system.
Sensors: GPS	Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system.
Sensors: Proximity Sensor	Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen.
Default Skin	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the resolution can mean that the screen is cut off, so you can't see the whole screen. See Create an emulator skin for more information.

AVD properties

You can specify the following properties for AVD configurations in the **Verify Configuration** page. The AVD configuration specifies the interaction between the development computer and the emulator, as well as properties you want to override in the hardware profile.

AVD configuration properties override hardware profile properties. Emulator properties that you set while the emulator is running override them both.

AVD Property	**Description**
AVD Name	Name of the AVD. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), dashes (-), and spaces. The name of the file storing the AVD configuration is derived from the AVD name.
AVD ID (Advanced)	The AVD filename is derived from the ID, and you can use the ID to refer to the AVD from the command line.
Hardware Profile	Click **Change** to select a different hardware profile in the **Select Hardware** page.
System Image	Click **Change** to select a different system image in the **System Image** page. An active internet connection is required to download a new image.
Startup Orientation	Select one option for the initial emulator orientation: • Portrait - Oriented taller than wide. • Landscape - Oriented wider than tall. An option is enabled only if it's selected in the hardware profile. When running the AVD in the emulator, you can change the orientation if both portrait and landscape are supported in the hardware profile.
Camera (Advanced)	To enable a camera, select one or both options: • Front - The lens faces away from the user. • Back - The lens faces toward the user. The **Emulated** setting produces a software-generated image, while the

	Webcam setting uses your development computer webcam to take a picture. This option is available only if it's selected in the hardware profile; it's not available for Wear OS and Android TV.
Network: Speed (Advanced)	Select a network protocol to determine the speed of data transfer: • GSM - Global System for Mobile Communications • HSCSD - High-Speed Circuit-Switched Data • GPRS - Generic Packet Radio Service • EDGE - Enhanced Data rates for GSM Evolution • UMTS - Universal Mobile Telecommunications System • HSDPA - High-Speed Downlink Packet Access • LTE - Long-Term Evolution • Full (default) - Transfer data as quickly as your computer allows.
Network: Latency (Advanced)	Select a network protocol to set how much time (delay) it takes for the protocol to transfer a data packet from one point to another point.
Emulated Performance: Graphics	Select how graphics are rendered in the emulator: • Hardware - Use your computer graphics card for faster rendering. • Software - Emulate the graphics in software, which is useful if you're having a problem with rendering in your graphics card. • Automatic - Let the emulator decide the best option based on your graphics card.
Emulated Performance: Boot option (Advanced)	• Cold boot - Start the device each time by powering up from the device-off state. • Quick boot - Start the device by loading the device state from a saved snapshot. For details, see Run the emulator with Quick Boot.
Emulated Performance: Multi-Core CPU (Advanced)	Select the number of processor cores on your computer that you'd like to use for the emulator. Using more processor cores speeds up the emulator.
Memory and Storage: RAM	The amount of RAM on the device. This value is set by the hardware manufacturer, but you can override it, if needed, such as for faster emulator operation. Increasing the size uses more resources on your computer. Type a RAM size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: VM Heap	The VM heap size. This value is set by the hardware manufacturer, but you can override it, if needed. Type a heap size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). For more information on Android VMs, see Memory Management for Different Virtual Machines.
Memory and Storage: Internal Storage	The amount of nonremovable memory space available on the device. This value is set by the hardware manufacturer, but you can override it, if needed. Type a size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: SD Card	The amount of removable memory space available to store data on the device. To use a virtual SD card managed by Android Studio, select **Studio-managed**, type a size, and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). A

	minimum of 100 MB is recommended to use the camera. To manage the space in a file, select **External file** and click **...** to specify the file and location. For more information, see mksdcard and AVD data directory.
Device Frame: Enable Device Frame	Select to enable a frame around the emulator window that mimics the look of a real device.
Custom Skin Definition (Advanced)	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the skin can mean that the screen is cut off, so you can't see the whole screen. See Create an emulator skin for more information.
Keyboard: Enable Keyboard Input (Advanced)	Select this option if you want to use your hardware keyboard to interact with the emulator. It's disabled for Wear OS and Android TV.

Create an emulator skin

An Android emulator skin is a collection of files that define the visual and control elements of an emulator display. If the skin definitions available in the AVD settings don't meet your requirements, you can create your own custom skin definition, and then apply it to your AVD.

Each emulator skin contains:

- A hardware.ini file
- Layout files for supported orientations (landscape, portrait) and physical configuration
- Image files for display elements, such as background, keys and buttons

To create and use a custom skin:

1. Create a new directory where you will save your skin configuration files.
2. Define the visual appearance of the skin in a text file named layout. This file defines many characteristics of the skin, such as the size and image assets for specific buttons. For example:

```
parts {
   device {
      display {
         width   320
         height  480
         x       0
         y       0
      }
   }

portrait {
   background {
      image background_port.png
   }

   buttons {
      power {
         image  button_vertical.png
         x 1229
         y 616
```

```
            }
        }
    }
    ...
    }
```

3. Add the bitmap files of the device images in the same directory.
4. Specify additional hardware-specific device configurations in a hardware.ini file for the device settings, such as hw.keyboard and hw.lcd.density.
5. Archive the files in the skin folder and select the archive file as a custom skin.

System requirements for installing Android Studio

Windows

- **Microsoft® Windows® 7**/8/10 (32- or 64-bit)
- **3 GB RAM minimum**, 8 GB RAM recommended; **plus 1 GB for the Android Emulator**
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Requirements and recommendations for installing the emulator

To install the Android Emulator, select the **Android Emulator** component in the **SDK Tools** tab of the **SDK Manager**.

The Android Emulator has additional requirements beyond the basic system requirements for Android Studio:

- **SDK Tools 26.1.1 or higher**
- **64-bit processor**
- **Windows: CPU with UG (unrestricted guest) support**
- **HAXM 6.2.1** or later (HAXM 7.2.0 or later recommended)

The use of hardware acceleration has additional requirements on Windows and Linux:

- **Intel processor on Windows or Linux: Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality**
- AMD processor on Linux: AMD processor with support for AMD Virtualization (AMD-V) and Supplemental Streaming SIMD Extensions 3 (SSSE3)
- AMD processor on Windows: Android Studio 3.2 or higher and Windows 10 April 2018 release or higher for Windows Hypervisor Platform (WHPX) functionality

In addition, **Virtual Technology must be enabled in BIOS Setup to Load the Emulator.**

To work with Android 8.1 (API level 27) and higher system images, an attached webcam must have the capability to capture 720p frames.

Ex. No. 02.
Date:

Displaying “Welcome to Android Laboratory”

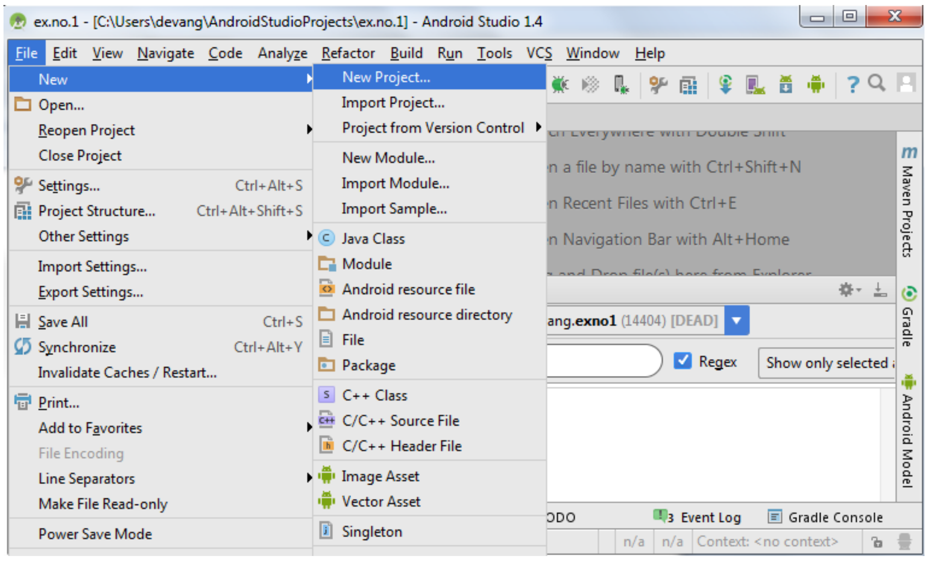
Aim:

To develop a Simple Android Application that displays a text.

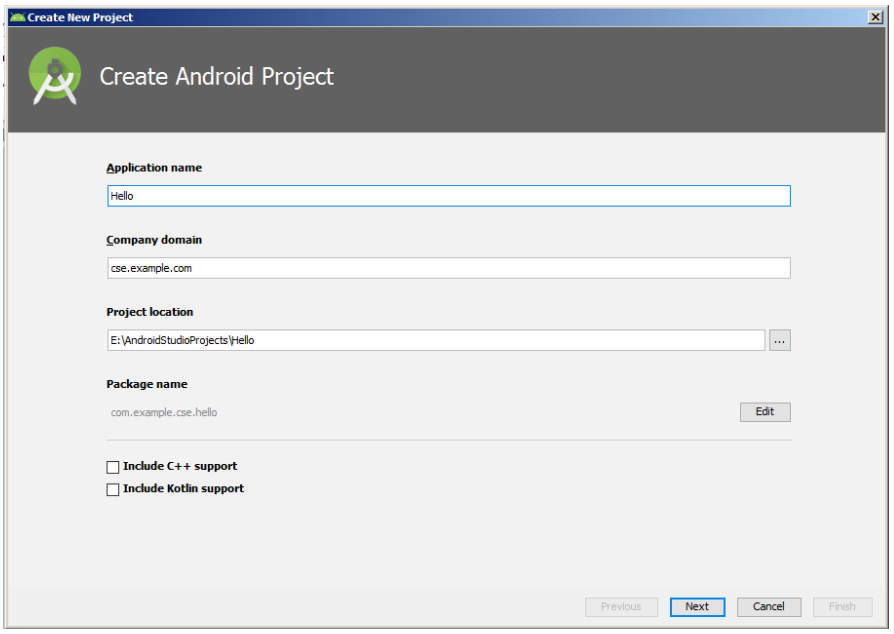
Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**



- Then type the Application name as “**Hello″**, change the project location and click **Next.**


Create New Project

Create Android Project

Application name
Hello

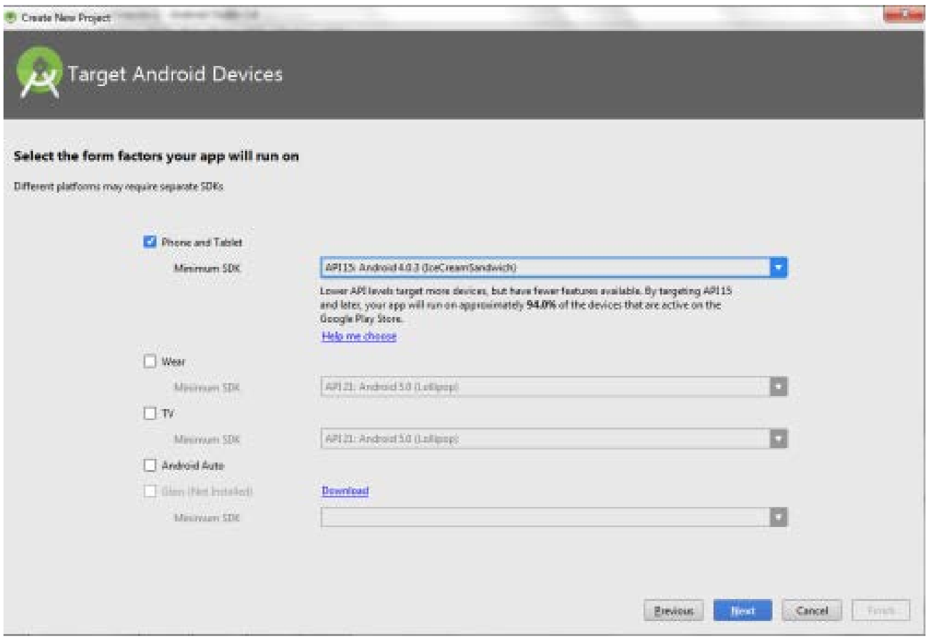
Company domain
cse.example.com

Project location
E:\AndroidStudioProjects\Hello

Package name
com.example.cse.hello

☐ Include C++ support
☐ Include Kotlin support

- Then select the **Minimum SDK** as shown below and click **Next.**


Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately **94.0%** of the devices that are active on the Google Play Store.

Help me choose

[ ] Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] TV

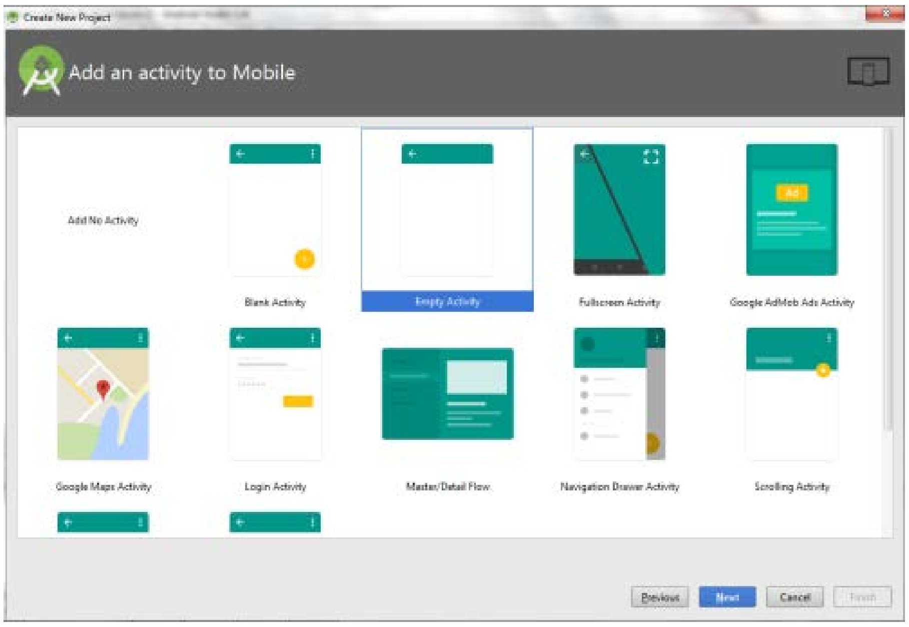
Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] Android Auto

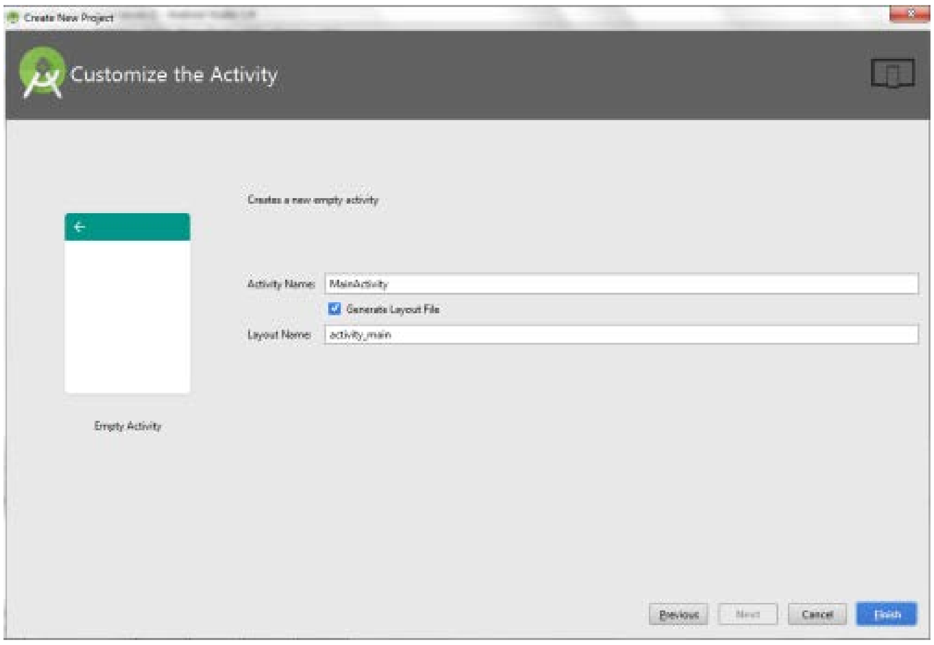
[ ] Glass (Not Installed) Download

Minimum SDK

Then select the **Empty Activity** and click **Next.**



- Finally click **Finish**.



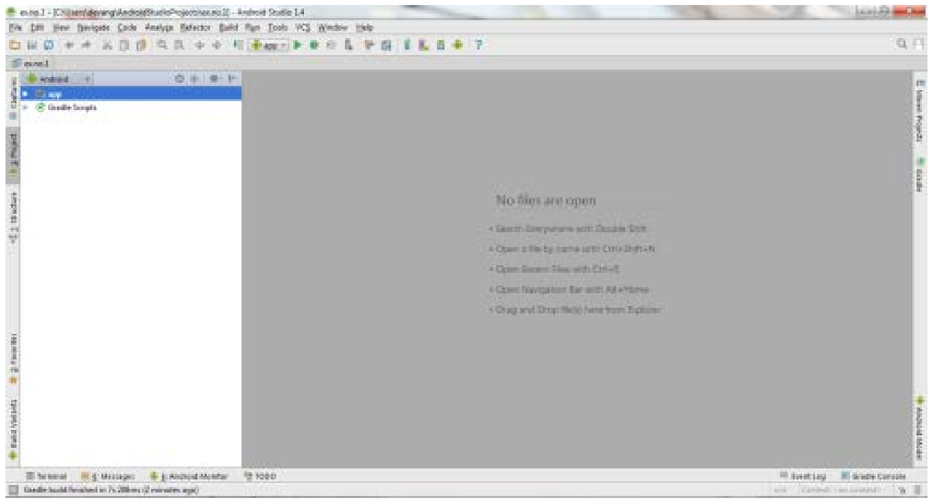
Creates a new empty activity

Activity Name: MainActivity

[✓] Generate Layout File

Layout Name: activity_main

- It will take some time to build and load the project.
- After completion it will look as given below.



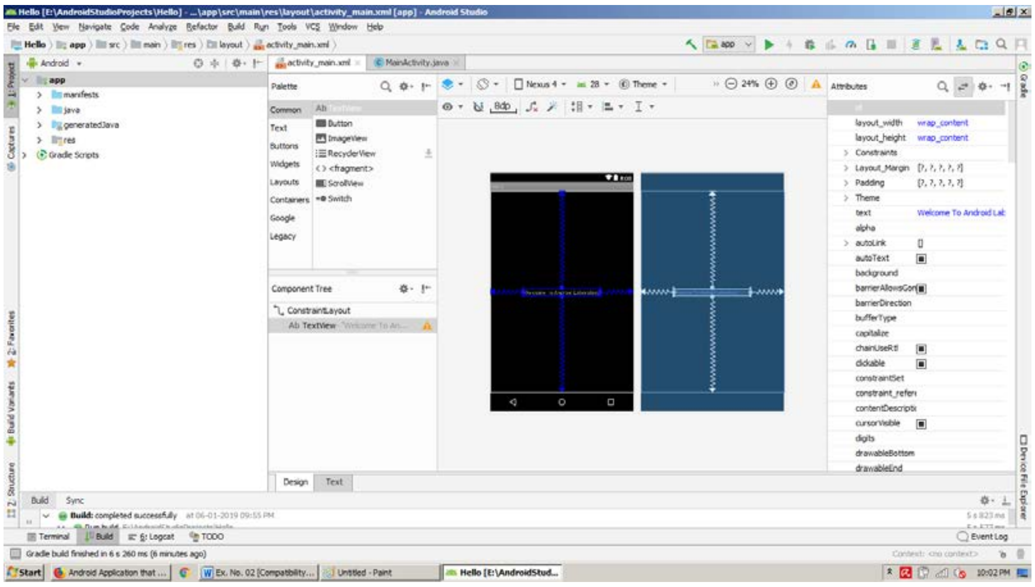
No files are open

- Search Everywhere with Double Shift
- Open a file by name with Ctrl+Shift+N
- Open Recent Files with Ctrl+E
- Open Navigation Bar with Alt+Home
- Drag and Drop file(s) here from Explorer

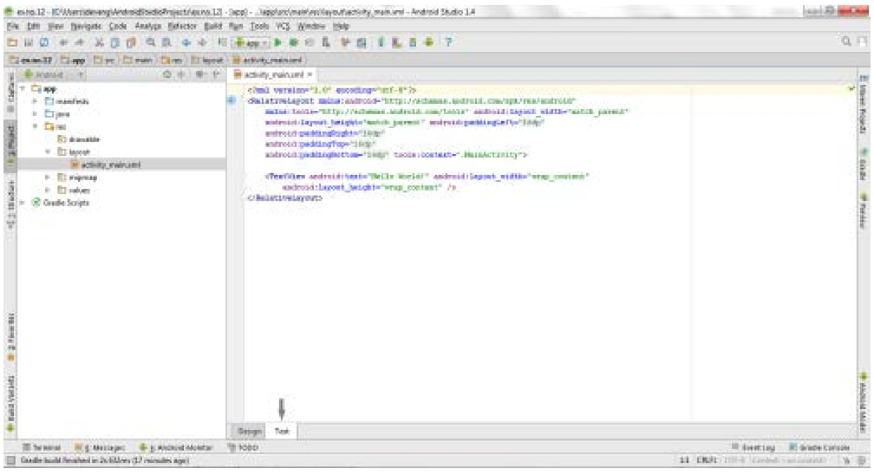
Gradle build finished in 7s 28ms (2 minutes ago)

Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Under design tab click the default text at the centre and select the text and locate the text values under the attribute window show at the right side.
- Change the text from “Hello world!” to your desired text “Welcome to Android Laboratory**”**

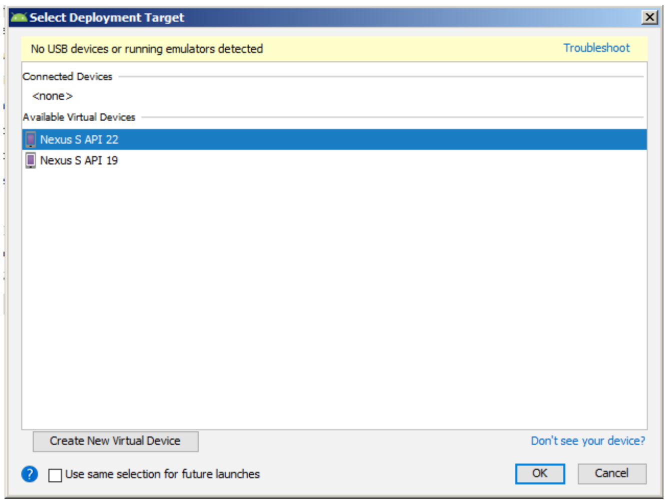


- Now click on **Text** as shown below.
- If needed change the below java code to the requirement.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="Hello World!" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Click Play icon or Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.



Select Deployment Target

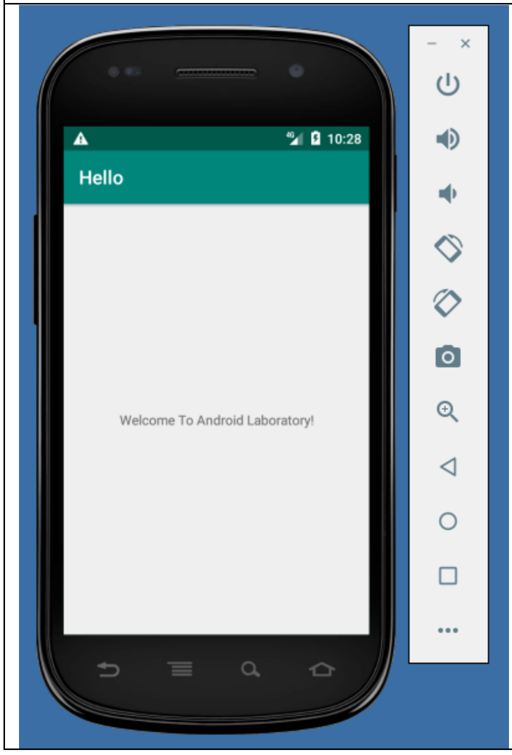
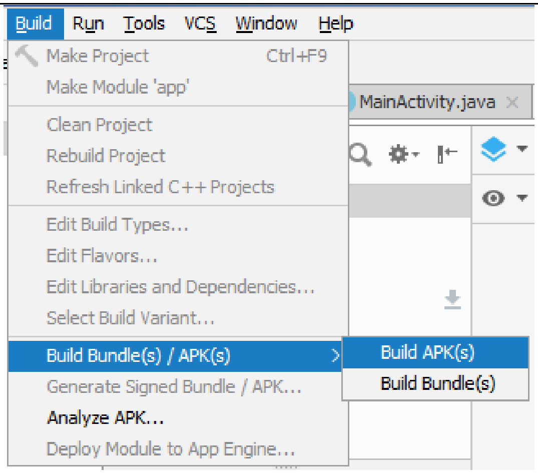
No USB devices or running emulators detected — Troubleshoot

Connected Devices
<none>

Available Virtual Devices
Nexus S API 22
Nexus S API 19

Create New Virtual Device — Don't see your device?

Use same selection for future launches — OK — Cancel

Emulator will be loaded and display the output of the app developed as show below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
	

Result:

Thus a Simple Android Application that displays a text is developed and executed successfully in emulator and Mobile device.

Ex. No. 03.
Date:

Designing Simple Toast

Aim:

To develop a Simple Toast application.

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as “Toast App″, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click **Finish**.

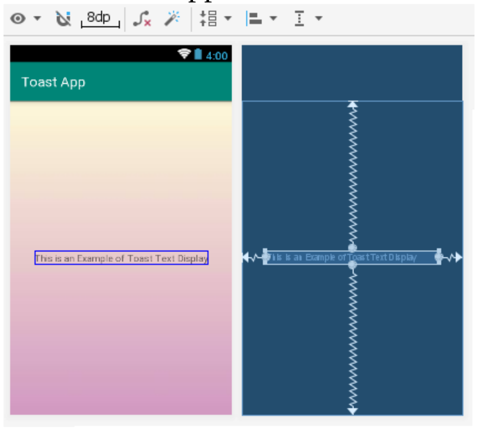
Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** shown below.
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/images1"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is an Example of Toast Text Display"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Note : Copy and paste the background image1.png under res/drawable folder

Now click on the Design tab and now the application will look as given below.

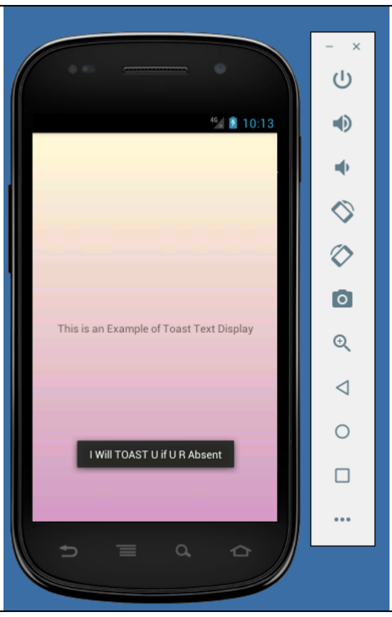
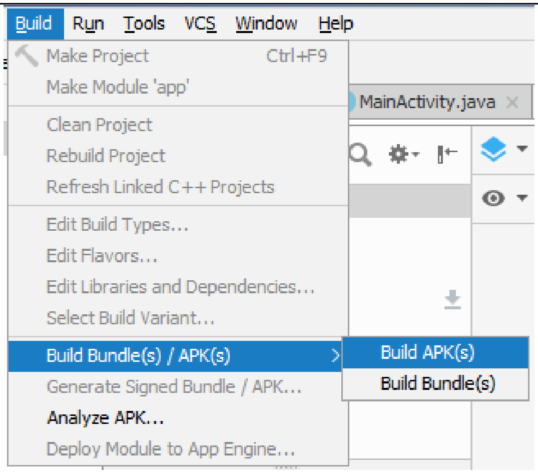


Java Coding for the Android Application:

- Click on **app -> java -> com.example. toastapp -> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.toastapp;

import android.os.Bundle;
import android.app.Activity;
import android.widget.Toast;

public class MainActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      // Displaying Toast with 'Studytonight is Best' message
      Toast.makeText(getApplicationContext(),"I Will TOAST U if U R Absent",
                                    Toast.LENGTH_LONG).show();
   }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
 10:13 This is an Example of Toast Text Display I Will TOAST U if U R Absent	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... Build APK(s) Build Bundle(s) MainActivity.java

Result:

Thus a Simple Android Application to display the Toast message is developed and executed successfully in emulator and Mobile device.

Ex. No. 04.
Date:

Designing User Interface based on Layouts

Aim:

To develop a Simple Android Application to design a User Interface based on Layouts.

Procedure:

Creating a New project:

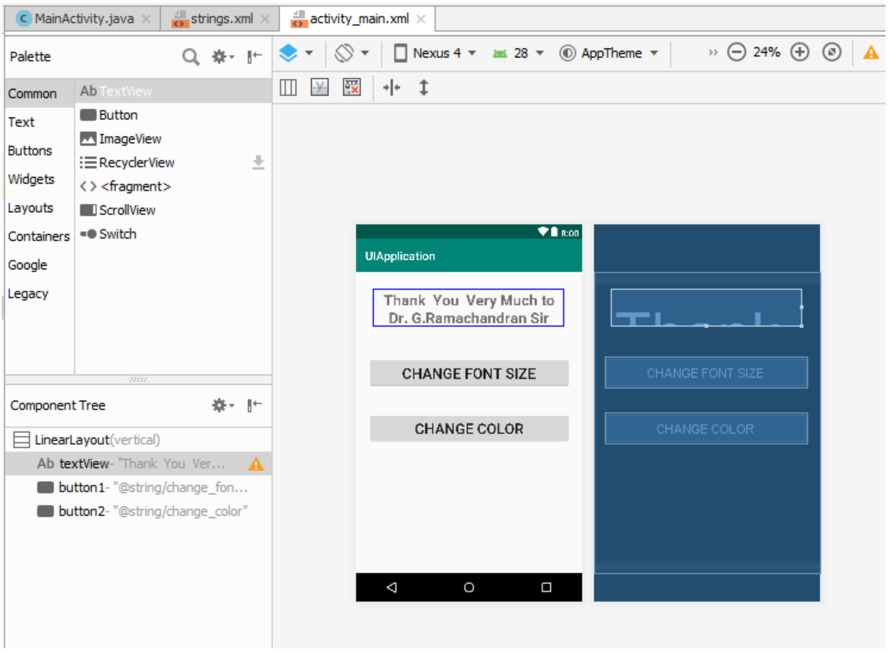
- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as "UIApplication″, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click F**inish**.

Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** as shown below.
- Then delete the code which is there and type the code as given below.

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="30dp"
        android:gravity="center"
        android:text="Thank  You  Very Much to Dr. G.Ramachandran Sir"
        android:textSize="25sp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:gravity="center"
        android:text="@string/change_font_size"
        android:textSize="25sp" />
    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:gravity="center"
        android:text="@string/change_color"
        android:textSize="25sp" />
</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.

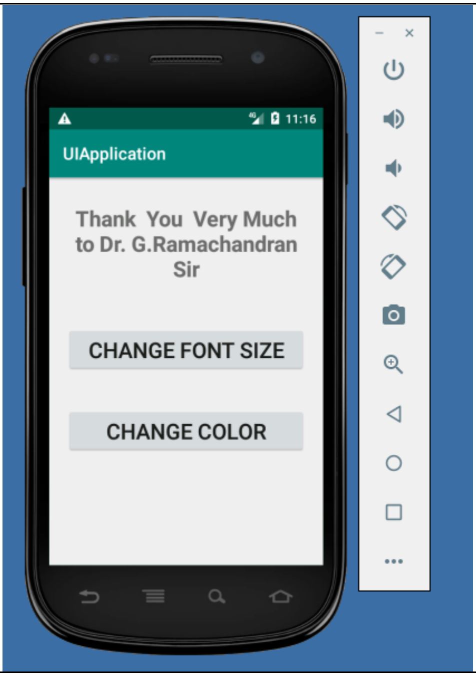
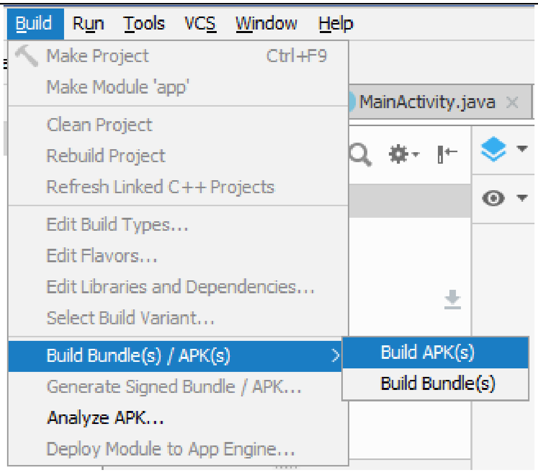


Java Coding for the Android Application:

- Click on **app -> java -> com.example.exno1 -> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.uiapplication;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity
{
    int ch=1;
    float font=30;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView t= (TextView) findViewById(R.id.textView);
        Button b1= (Button) findViewById(R.id.button1);
        b1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                t.setTextSize(font);
                font = font + 5;
                if (font == 50)
                    font = 30;
            }
        });
        Button b2= (Button) findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                switch (ch) {
                    case 1:
                        t.setTextColor(Color.RED);
                        break;
                    case 2:
                        t.setTextColor(Color.GREEN);
                        break;
                    case 3:
                        t.setTextColor(Color.BLUE);
                        break;
                    case 4:
                        t.setTextColor(Color.CYAN);
                        break;
                    case 5:
                        t.setTextColor(Color.YELLOW);
                        break;
                    case 6:
                        t.setTextColor(Color.MAGENTA);
                        break;
                }
                ch++;
                if (ch == 7)
                    ch = 1;
            }
        });
    }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
 11:16 UIApplication Thank You Very Much to Dr. G.Ramachandran Sir CHANGE FONT SIZE CHANGE COLOR	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Build APK(s) Build Bundle(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... MainActivity.java

Result:

Thus a Simple Android Application to design a User Interface based on Layouts is developed and executed successfully in emulator and Mobile device.

Ex. No. 05.
Date:

Displaying different Shapes

Aim:

To develop a Simple Android Application to display different shapes.

Procedure:

Creating a New project:

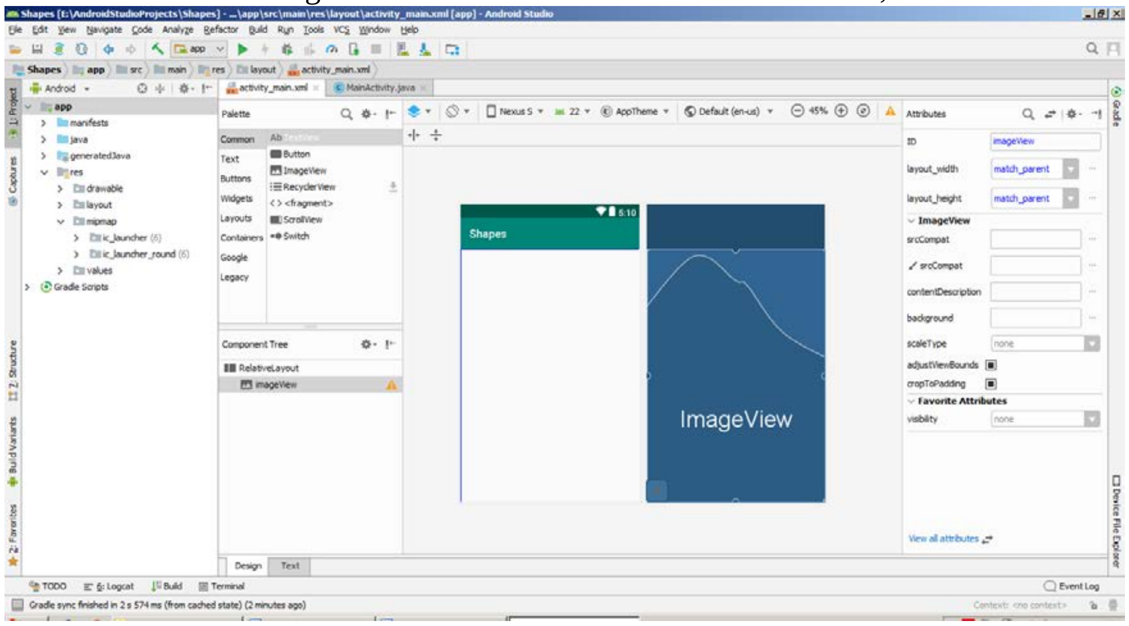
- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as "**Shapes**", change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click F**inish**.

Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Delete the text there and type the below xml code.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageView" />
</RelativeLayout>
```

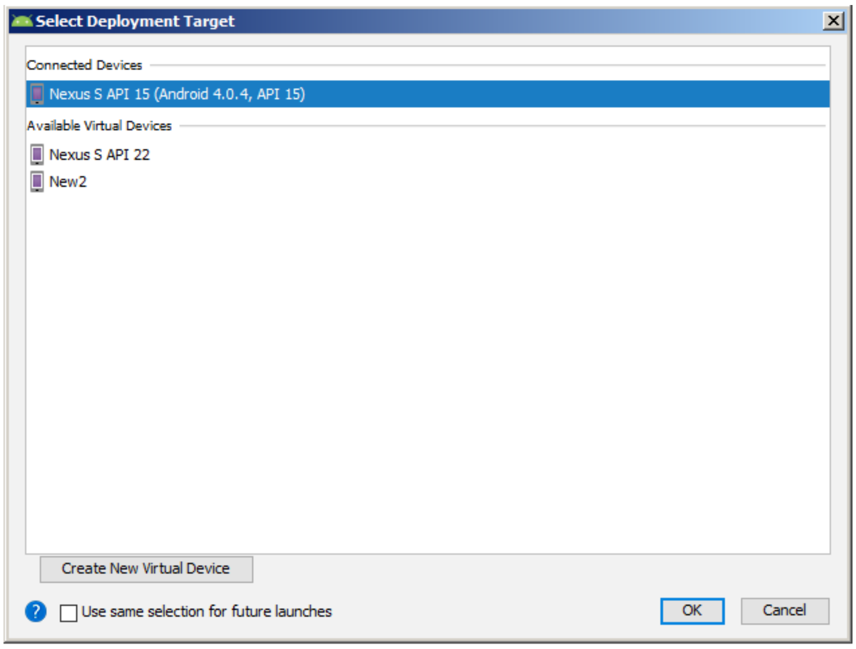
- Now click the Design tab and the screen will look like as shown,


Shapes [E:\AndroidStudioProjects\Shapes] - ...\app\src\main\res\layout\activity_main.xml [app] - Android Studio
Palette
Component Tree
RelativeLayout
imageView
Shapes
ImageView
Attributes
ID
imageView
layout_width
match_parent
layout_height
match_parent
Design
Text
Gradle sync finished in 2 s 574 ms (from cached state) (2 minutes ago)

- Now click on **MainActivity.java** shown below.
- Delete the system generated java code and type the below code

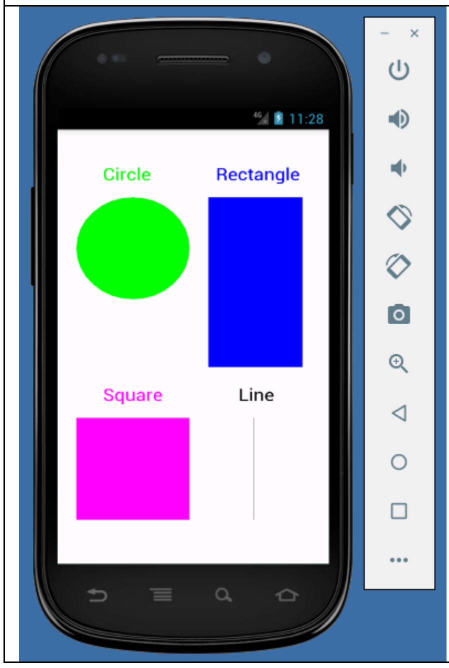
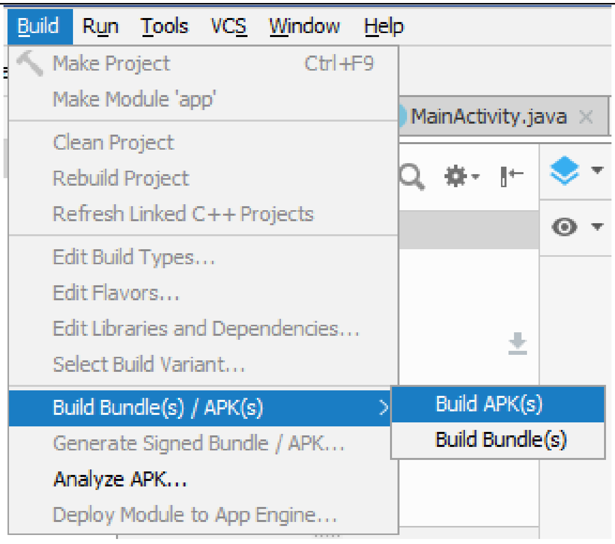
```
package com.example.cse.shapes;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      //Creating a Bitmap
      Bitmap bg = Bitmap.createBitmap(720, 1280, Bitmap.Config.ARGB_8888);

      //Setting the Bitmap as background for the ImageView
      ImageView i = (ImageView) findViewById(R.id.imageView);
      i.setBackgroundDrawable(new BitmapDrawable(bg));

      //Creating the Canvas Object
      Canvas canvas = new Canvas(bg);

      //Creating the Paint Object and set its color & TextSize
      Paint paint = new Paint();
      paint.setColor(Color.BLUE);
      paint.setTextSize(50);

      //To draw a Rectangle
      canvas.drawText("Rectangle", 420, 150, paint);
      canvas.drawRect(400, 200, 650, 700, paint);
      paint.setColor(Color.GREEN);
      //To draw a Circle
      canvas.drawText("Circle", 120, 150, paint);
      canvas.drawCircle(200, 350, 150, paint);
      paint.setColor(Color.MAGENTA);

      //To draw a Square
      canvas.drawText("Square", 120, 800, paint);
      canvas.drawRect(50, 850, 350, 1150, paint);
      paint.setColor(Color.BLACK);
      //To draw a Line
      canvas.drawText("Line", 480, 800, paint);
      canvas.drawLine(520, 850, 520, 1150, paint);
   }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.


Select Deployment Target

Connected Devices
Nexus S API 15 (Android 4.0.4, API 15)
Available Virtual Devices
Nexus S API 22
New2

Create New Virtual Device
Use same selection for future launches
OK
Cancel

Emulator will be loaded and display the output of the app developed as show below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... Build APK(s) Build Bundle(s) MainActivity.java

Result:

Thus a Simple Android Application that displays different shapes is developed and executed successfully in emulator and Mobile device.

Ex. No. 06.
Date:

Designing Simple Calculator Application

Aim:

To develop a Simple Android Application to design a Simple Calculator App.

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as “CalcApp″, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click F**inish**.

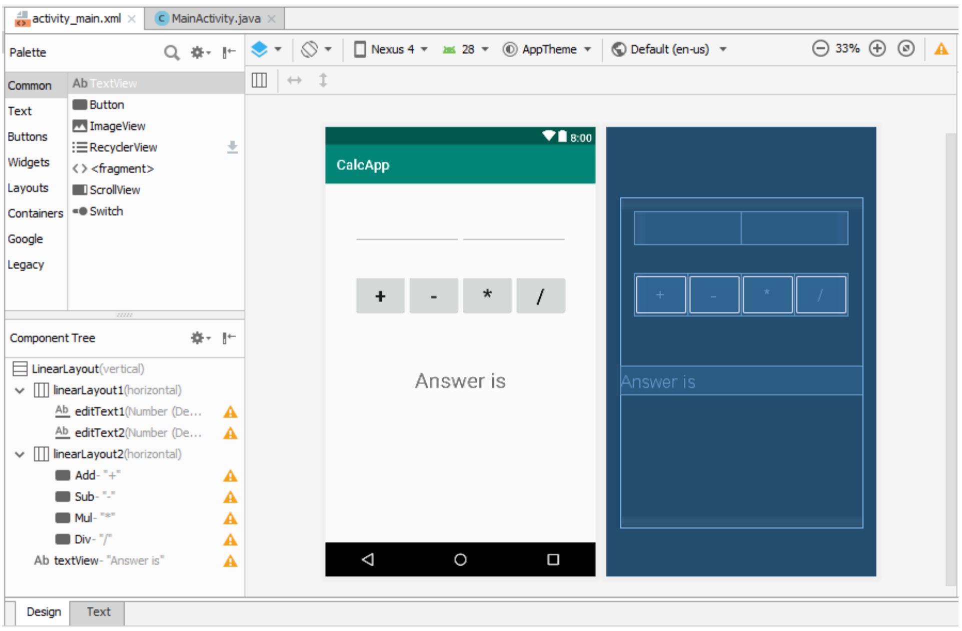
Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** shown below.
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="20dp">

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp">

        <EditText
            android:id="@+id/editText1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="numberDecimal"
            android:textSize="20sp" />

        <EditText
            android:id="@+id/editText2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="numberDecimal"
            android:textSize="20sp" />

    </LinearLayout>

    <LinearLayout
```

```
        android:id="@+id/linearLayout2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp">

        <Button
            android:id="@+id/Add"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="+"
            android:textSize="30sp"/>

        <Button
            android:id="@+id/Sub"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="-"
            android:textSize="30sp"/>

        <Button
            android:id="@+id/Mul"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="*"
            android:textSize="30sp"/>

        <Button
            android:id="@+id/Div"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="/"
            android:textSize="30sp"/>

    </LinearLayout>

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="Answer is"
        android:textSize="30sp"
        android:gravity="center"/>

</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.



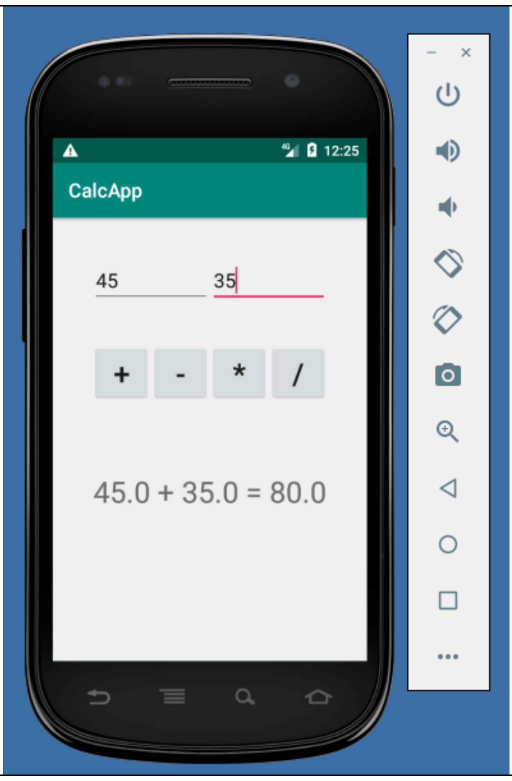
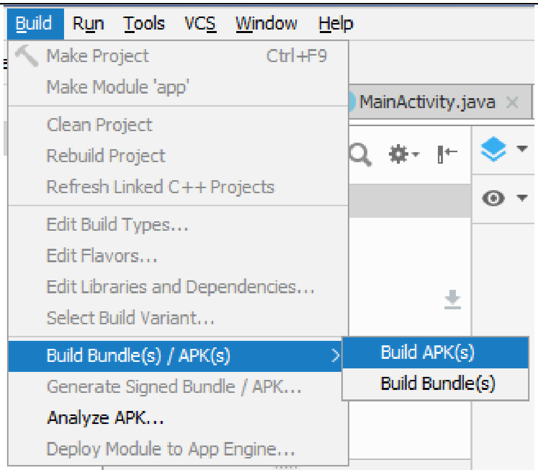
Java Coding for the Android Application:

- Click on **app -> java -> com.example. calcapp -> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.calcapp;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.TextUtils;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements OnClickListener
{
   //Defining the Views
   EditText Num1;
   EditText Num2;
   Button Add;
   Button Sub;
   Button Mul;
   Button Div;
   TextView Result;

   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      //Referring the Views
      Num1 = (EditText) findViewById(R.id.editText1);
      Num2 = (EditText) findViewById(R.id.editText2);
```

```
        Add = (Button) findViewById(R.id.Add);
        Sub = (Button) findViewById(R.id.Sub);
        Mul = (Button) findViewById(R.id.Mul);
        Div = (Button) findViewById(R.id.Div);
        Result = (TextView) findViewById(R.id.textView);

        // set a listener
        Add.setOnClickListener(this);
        Sub.setOnClickListener(this);
        Mul.setOnClickListener(this);
        Div.setOnClickListener(this);
    }

    @Override
    public void onClick (View v)
    {

        float num1 = 0;
        float num2 = 0;
        float result = 0;
        String oper = "";
        // check if the fields are empty
        if (TextUtils.isEmpty(Num1.getText().toString()) ||
TextUtils.isEmpty(Num2.getText().toString()))
            return;

        // read EditText and fill variables with numbers
        num1 = Float.parseFloat(Num1.getText().toString());
        num2 = Float.parseFloat(Num2.getText().toString());

        // defines the button that has been clicked and performs the corresponding operation
        // write operation into oper, we will use it later for output
        switch (v.getId())
        {
            case R.id.Add:
                oper = "+";
                result = num1 + num2;
                break;
            case R.id.Sub:
                oper = "-";
                result = num1 - num2;
                break;
            case R.id.Mul:
                oper = "*";
                result = num1 * num2;
                break;
            case R.id.Div:
                oper = "/";
                result = num1 / num2;
                break;
            default:
                break;
        }
        // form the output line
        Result.setText(num1 + " " + oper + " " + num2 + " = " + result);
    }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
 12:25 CalcApp 45 35 + - * / 45.0 + 35.0 = 80.0	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... Build APK(s) Build Bundle(s) MainActivity.java

Result:

Thus a Simple Android Application to design a Simple Calculator App is developed and executed successfully in emulator and Mobile device.

Ex. No. 07.
Date:

Navigation in Android

Aim:

To develop a Simple Android Application to design a simple Navigation between two activities.

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as “NavigationApp″, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- **Creating Second Activity for the Android Application:**
- Click on **File -> New -> Activity -> Empty Activity.**
- Finally click F**inish**.

Designing layout for the Android Application:

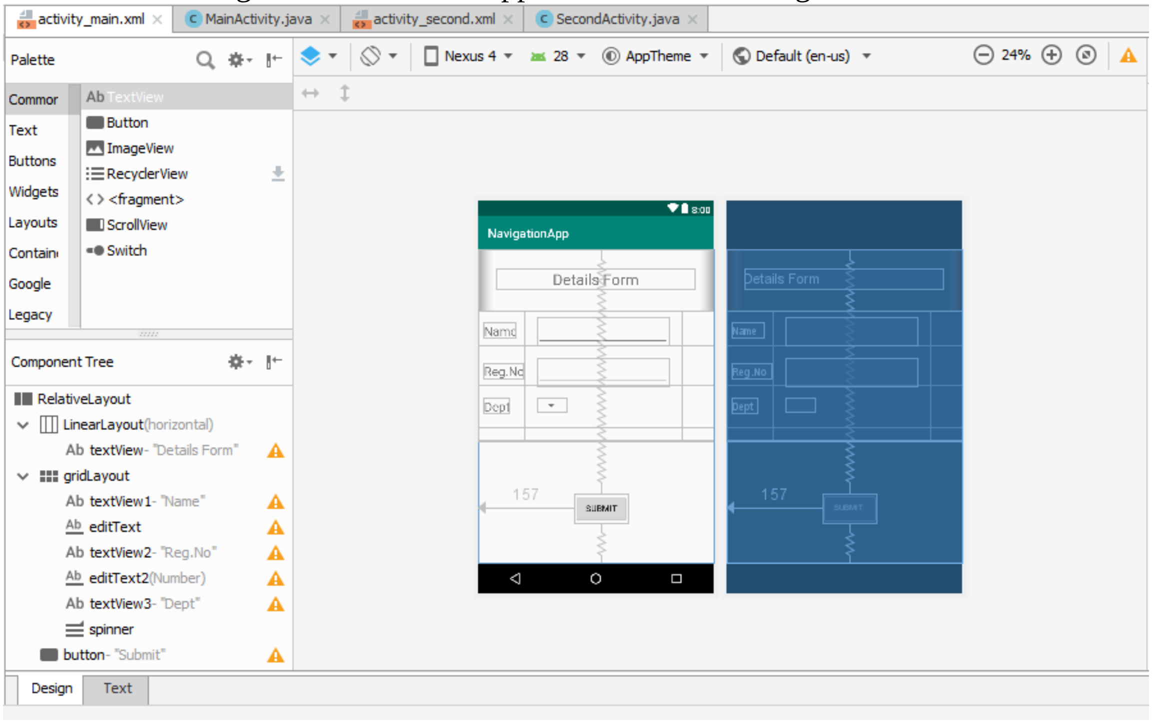
- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** as shown below.
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="100dp">
        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="30dp"
            android:text="Details Form"
            android:textSize="25sp"
            android:gravity="center"/>
    </LinearLayout>

    <GridLayout
        android:id="@+id/gridLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="100dp"
```

```
android:layout_marginBottom="200dp"
android:columnCount="2"
android:rowCount="3">
<TextView
   android:id="@+id/textView1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="10dp"
   android:layout_row="0"
   android:layout_column="0"
   android:text="Name"
   android:textSize="20sp"
   android:gravity="center"/>

<EditText
   android:id="@+id/editText"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="10dp"
   android:layout_row="0"
   android:layout_column="1"
   android:ems="10"/>

<TextView
   android:id="@+id/textView2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="10dp"
   android:layout_row="1"
   android:layout_column="0"
   android:text="Reg.No"
   android:textSize="20sp"
   android:gravity="center"/>

<EditText
   android:id="@+id/editText2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="10dp"
   android:layout_row="1"
   android:layout_column="1"
   android:inputType="number"
   android:ems="10"/>

<TextView
   android:id="@+id/textView3"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="10dp"
   android:layout_row="2"
   android:layout_column="0"
```

```
            android:text="Dept"
            android:textSize="20sp"
            android:gravity="center"/>

        <Spinner
            android:id="@+id/spinner"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_row="2"
            android:layout_column="1"
            android:spinnerMode="dropdown"/>

    </GridLayout>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true"
        android:layout_centerVertical="true"
        android:layout_marginStart="157dp"
        android:layout_marginLeft="157dp"
        android:layout_marginBottom="66dp"
        android:text="Submit" />

</RelativeLayout>
```

Now click on the Design tab and now the application will look as given below.

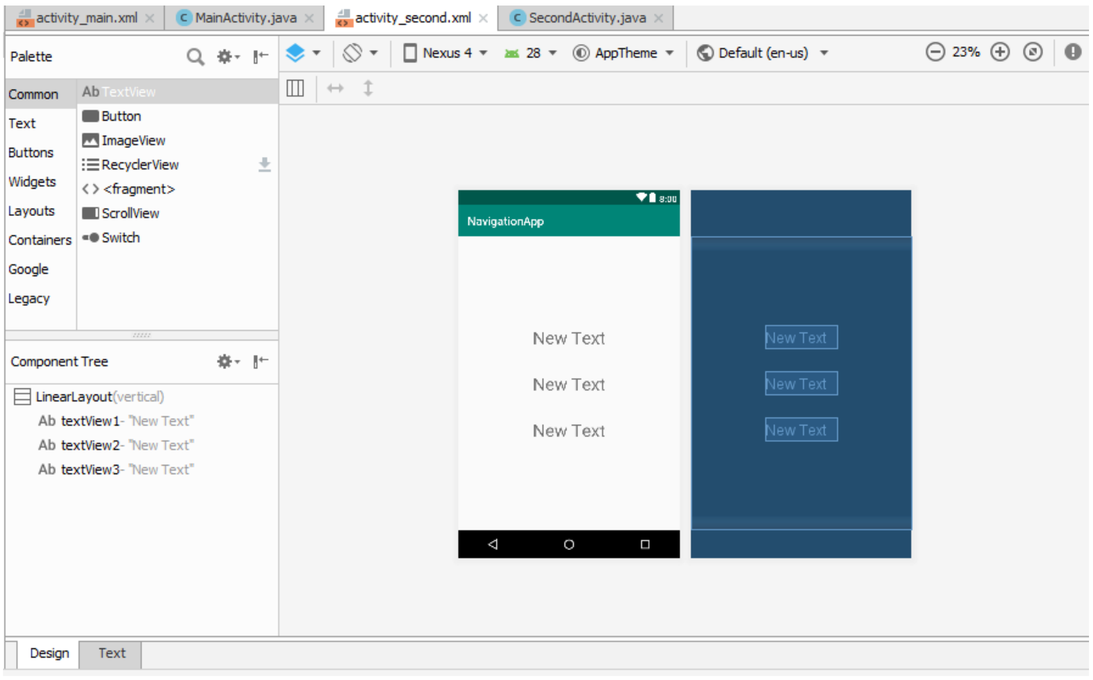


Designing Layout for Second Activity:

- Click on **app -> res -> layout -> activity_second.xml.**
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.cse.navigationapp.SecondActivity"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:text="New Text"
        android:textSize="30sp"/>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:text="New Text"
        android:textSize="30sp"/>

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:text="New Text"
        android:textSize="30sp"/>

</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.



Java Coding for the Android Application:

Java Coidng for Main Activity:

- Click on **app -> java -> com.example.**navigationapp **-> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.navigationapp;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;

public class MainActivity extends AppCompatActivity {

    //Defining the Views
    EditText e1,e2;
    Button bt;
    Spinner s;

    //Data for populating in Spinner
    String [] dept_array={"CSE","ECE","Mech","Civil"};

    String name,reg,dept;

    @Override
```

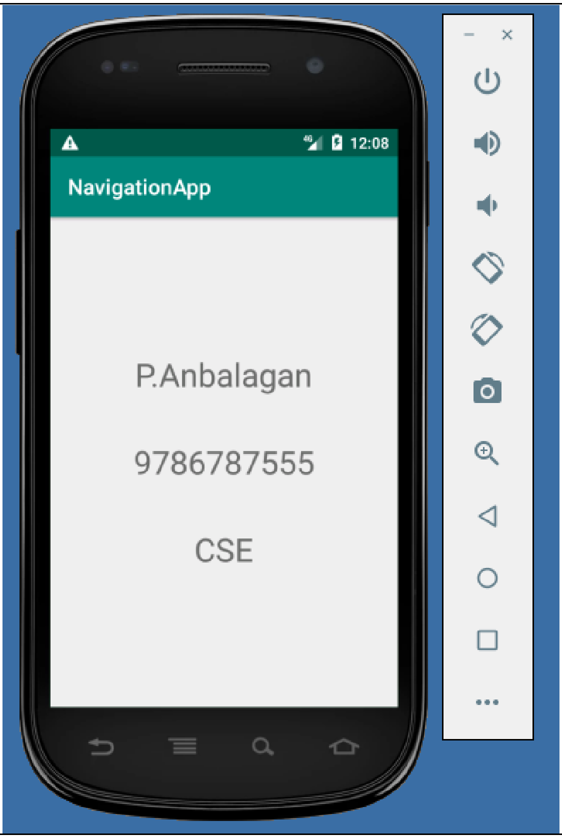
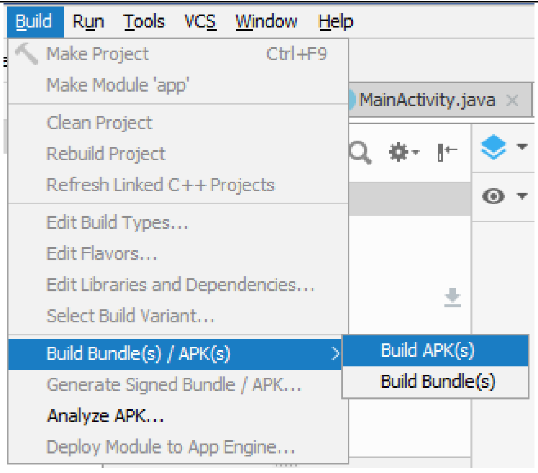
```
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //Referring the Views
    e1= (EditText) findViewById(R.id.editText);
    e2= (EditText) findViewById(R.id.editText2);

    bt= (Button) findViewById(R.id.button);

    s= (Spinner) findViewById(R.id.spinner);

    //Creating Adapter for Spinner for adapting the data from array to Spinner
    ArrayAdapter adapter= new
ArrayAdapter(MainActivity.this,android.R.layout.simple_spinner_item,dept_array);
    s.setAdapter(adapter);

    //Creating Listener for Button
    bt.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {

        //Getting the Values from Views(Edittext & Spinner)
        name=e1.getText().toString();
        reg=e2.getText().toString();
        dept=s.getSelectedItem().toString();

        //Intent For Navigating to Second Activity
        Intent i = new Intent(MainActivity.this,SecondActivity.class);

        //For Passing the Values to Second Activity
        i.putExtra("name_key", name);
        i.putExtra("reg_key",reg);
        i.putExtra("dept_key", dept);

        startActivity(i);

      }
    });
  }
}
```

Java Coding for Second Activity:

- Click on **app -> java -> com.example.** navigationapp **-> SecondActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.navigationapp;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {

    TextView t1,t2,t3;

    String name,reg,dept;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);

        t1= (TextView) findViewById(R.id.textView1);
        t2= (TextView) findViewById(R.id.textView2);
        t3= (TextView) findViewById(R.id.textView3);

        //Getting the Intent
        Intent i = getIntent();

        //Getting the Values from First Activity using the Intent received
        name=i.getStringExtra("name_key");
        reg=i.getStringExtra("reg_key");
        dept=i.getStringExtra("dept_key");

        //Setting the Values to Intent
        t1.setText(name);
        t2.setText(reg);
        t3.setText(dept);

    }
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... Build APK(s) Build Bundle(s) MainActivity.java

Result:

Thus a Simple Android Application to Navigation between two activities is developed and executed successfully in emulator and Mobile device.

Ex. No. 08.
Date:

Displaying the Notification

Aim:

To develop a Simple Android Application to create and display the Notification

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as "**Notification″**, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Create the Second Activity with the name **"SecondAcitivity"**
- Click on **File -> New -> Activity -> Empty Activity.**
- Finally click F**inish**.

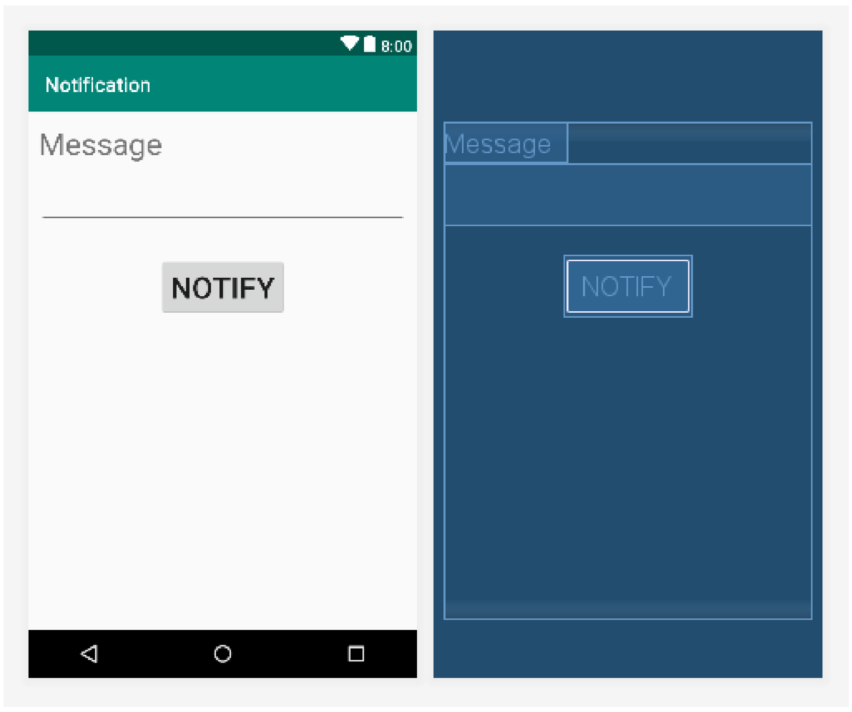
Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** as shown below.
- Then delete the code which is there and type the code as given below.

```
<?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:layout_margin="10dp"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message"
        android:textSize="30sp" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:textSize="30sp" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="30dp"
        android:layout_gravity="center"
        android:text="Notify"
        android:textSize="30sp"/>
```

```
</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.

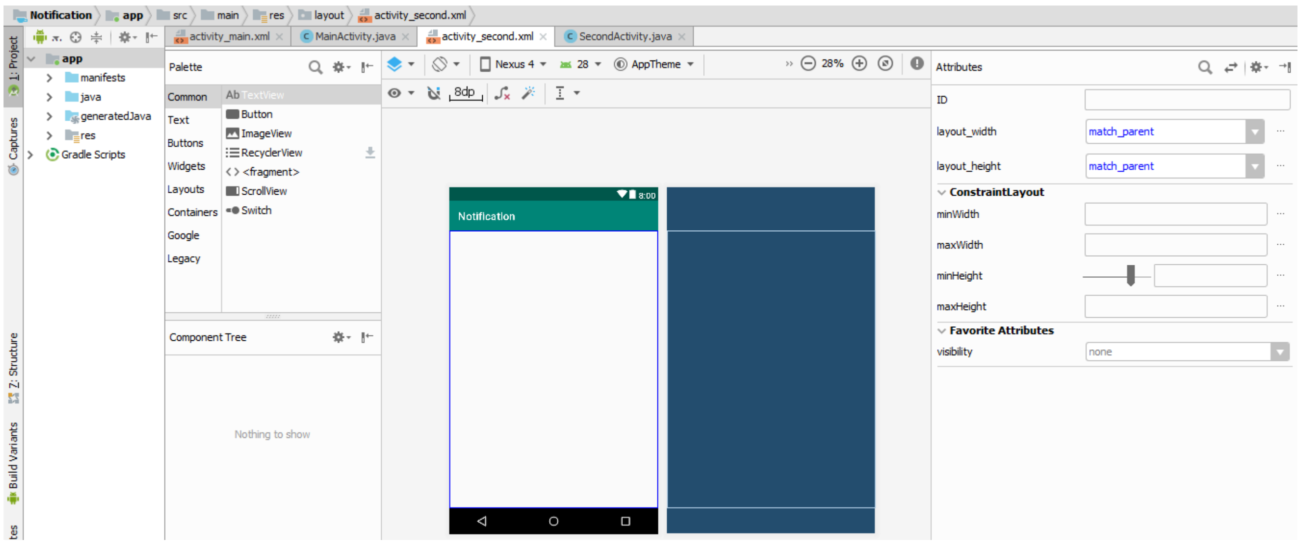


Designing Layout for Second Activity:

- Click on **app -> res -> layout -> acivity_second.xml.**
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

</android.support.constraint.ConstraintLayout>
```

Now click on the Design tab and now the application will look as given below.



Java Coding for the Android Application:

Java Coidng for Main Activity:

- Click on **app -> java -> com.example.**notification **-> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.notification;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity
{
    Button notify;
    EditText e;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        notify= (Button) findViewById(R.id.button);
        e= (EditText) findViewById(R.id.editText);

        notify.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
```

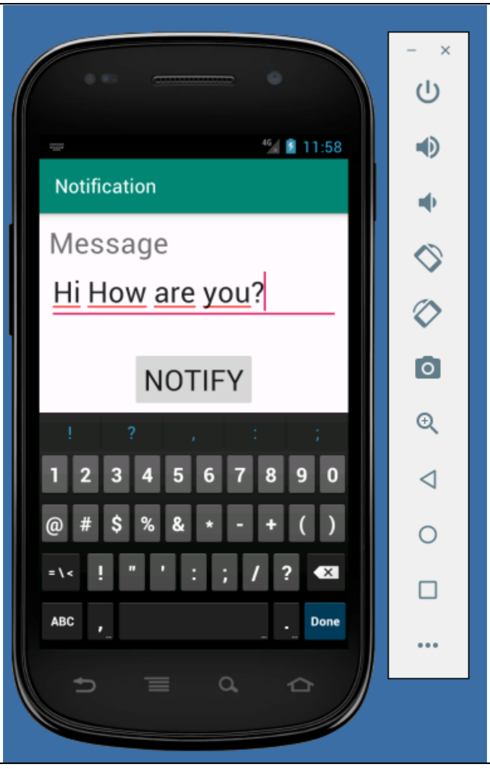
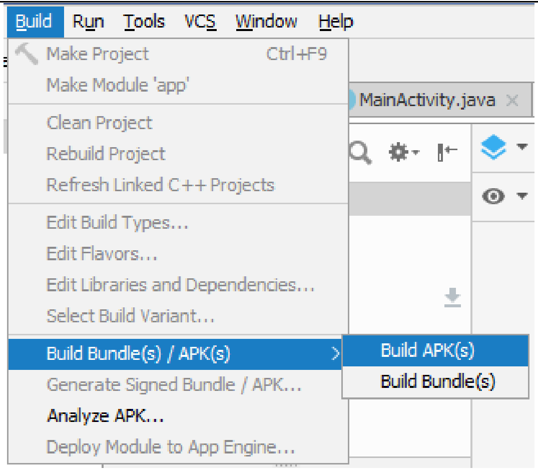
```
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);
            PendingIntent pending = PendingIntent.getActivity(MainActivity.this, 0, intent, 0);
            Notification noti = new Notification.Builder(MainActivity.this).setContentTitle("New
Message").setContentText(e.getText().toString()).setSmallIcon(R.mipmap.ic_launcher).setContentInt
ent(pending).build();
            NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
            noti.flags |= Notification.FLAG_AUTO_CANCEL;
            manager.notify(0, noti);
         }
      });
   }
}
```

Java Coding for Second Activity:

- Click on **app -> java -> com.example.** notificaation **-> SecondActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.notification;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class SecondActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
   }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
	

Result:

Thus a Simple Android Application to display the notification is developed and executed successfully in emulator and Mobile device.

Ex. No. 09.
Date:

Creating an Alarm

Aim:

To develop a Simple Android Application to create and display the Notification

Procedure:

Creating a New project:

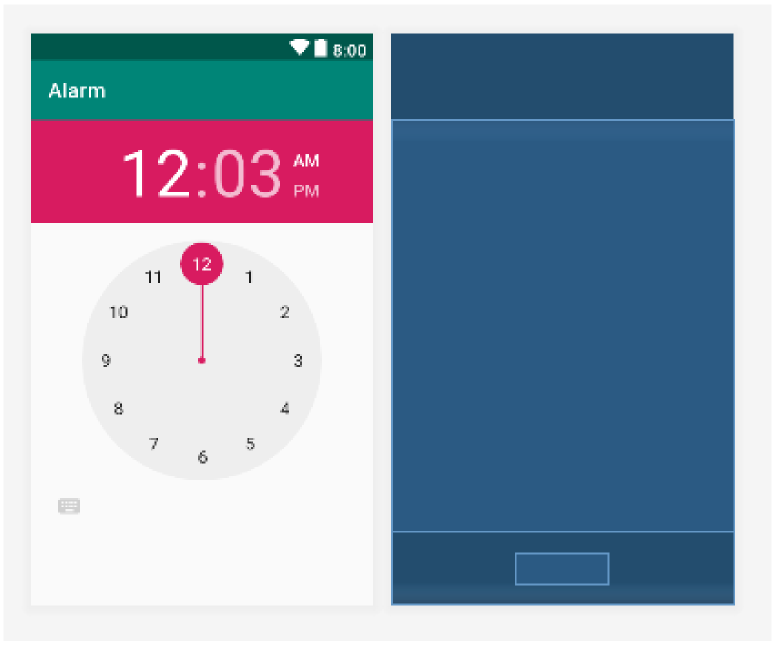
- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as “**Notification″**, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Create the Second Activity with the name **“activity_alarm_receiver”**
- Click on **File -> New -> Activity -> Empty Activity.**
- Finally click F**inish**.

Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** shown below.
- Then delete the code which is there and type the code as given below.

```
<?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">

   <TimePicker
      android:id="@+id/timePicker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center" />

   <ToggleButton
      android:id="@+id/toggleButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
      android:layout_margin="20dp"
      android:checked="false"
      android:onClick="OnToggleClicked" />

</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.

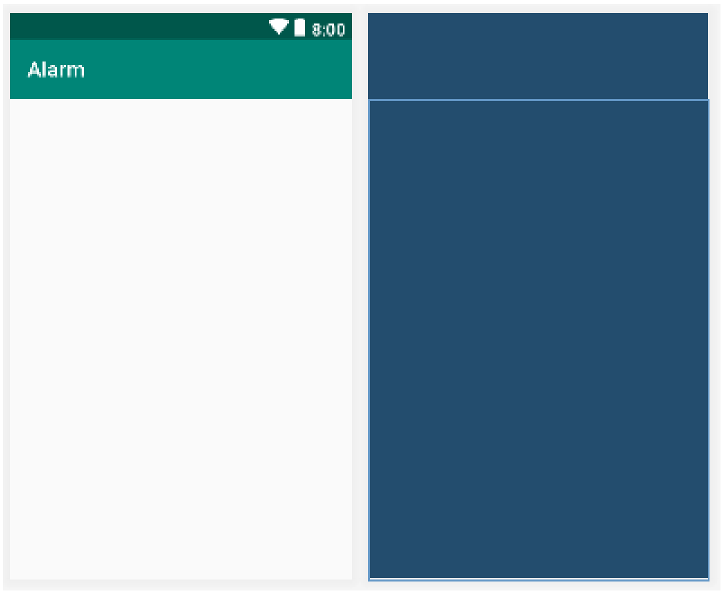


Designing Layout for Second Activity:

- Click on **app -> res -> layout -> activity_alarm_receiver.xml.**
- Then delete the code which is there and type the code as given below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AlarmReceiver">

</android.support.constraint.ConstraintLayout>
```

Now click on the Design tab and now the application will look as given below.



Java Coding for the Android Application:

Java Coidng for Main Activity:

- Click on **app -> java -> com.example.**notification **-> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package com.example.cse.alarm;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;
import android.widget.ToggleButton;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity
{
   TimePicker alarmTimePicker;
   PendingIntent pendingIntent;
   AlarmManager alarmManager;

   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      alarmTimePicker = (TimePicker) findViewById(R.id.timePicker);
      alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
   }
   public void OnToggleClicked(View view)
   {
      long time;
      if (((ToggleButton) view).isChecked())
      {
         Toast.makeText(MainActivity.this, "ALARM ON", Toast.LENGTH_SHORT).show();
         Calendar calendar = Calendar.getInstance();
         calendar.set(Calendar.HOUR_OF_DAY, alarmTimePicker.getCurrentHour());
         calendar.set(Calendar.MINUTE, alarmTimePicker.getCurrentMinute());
         Intent intent = new Intent(this, AlarmReceiver.class);
         pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

         time=(calendar.getTimeInMillis()-(calendar.getTimeInMillis()%60000));
         if(System.currentTimeMillis()>time)
         {
            if (calendar.AM_PM == 0)
               time = time + (1000*60*60*12);
            else
               time = time + (1000*60*60*24);
         }
```

```
            alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, time, 10000, pendingIntent);
        }
        else
        {
            alarmManager.cancel(pendingIntent);
            Toast.makeText(MainActivity.this, "ALARM OFF", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Java Coding for Second Activity:

- Click on **app -> java -> com.example.alarm -> alarmreceive.java.**
  (if it is not present, then create it as new)
- Then delete the code which is there and type the code as given below.

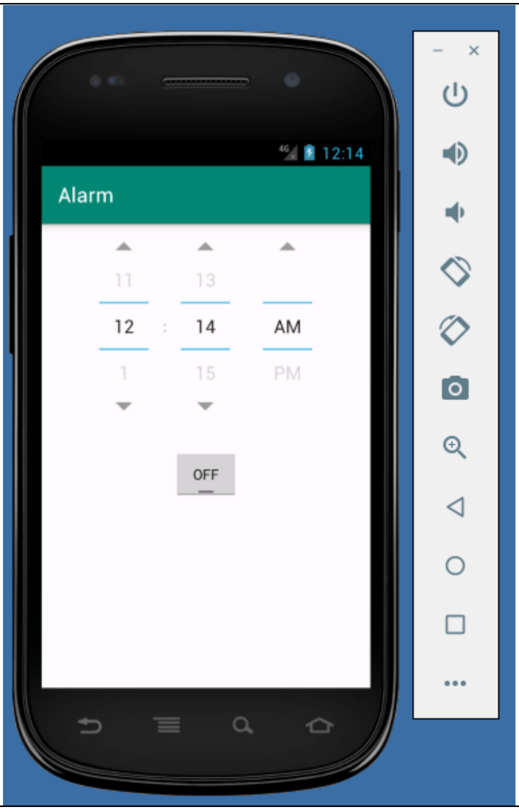
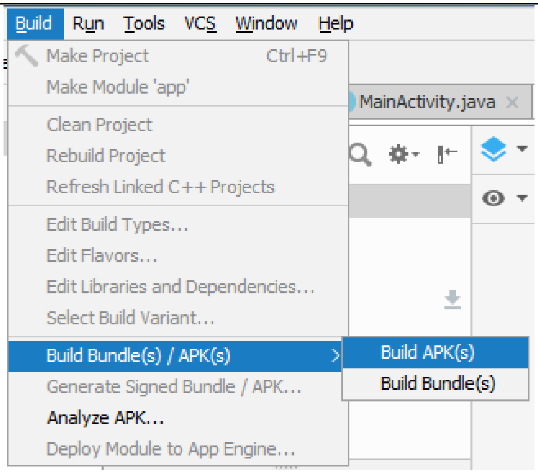
```
package com.example.cse.alarm;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.widget.Toast;

public class AlarmReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context, "Alarm! Wake up! Wake up!", Toast.LENGTH_LONG).show();
        Uri alarmUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
        if (alarmUri == null)
        {
            alarmUri =
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
        }
        Ringtone ringtone = RingtoneManager.getRingtone(context, alarmUri);
        ringtone.play();
    }
}
```

XML Coding for Android Manifest file:

- Click on **app -> manifests -> AndroidManifest.xml.**
- Then delete the code which is there and type the code as given below

```
<?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.cse.alarm" >

        <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:supportsRtl="true"
            android:theme="@style/AppTheme" >
            <activity android:name=".MainActivity" >
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />

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

    </manifest>
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below. (If the analog clock is not supported, then the digital clock will be displayed.)	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
	

Result:

Thus a Simple Android Application to simulate the alarm is developed and executed successfully in emulator and Mobile device.

Ex. No. 10.
Date:

Designing BMI Calculator Application

Aim:

To develop a Simple Android Application to design a Simple BMI Calculator

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as “BMICalc″, change the project location and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click F**inish**.

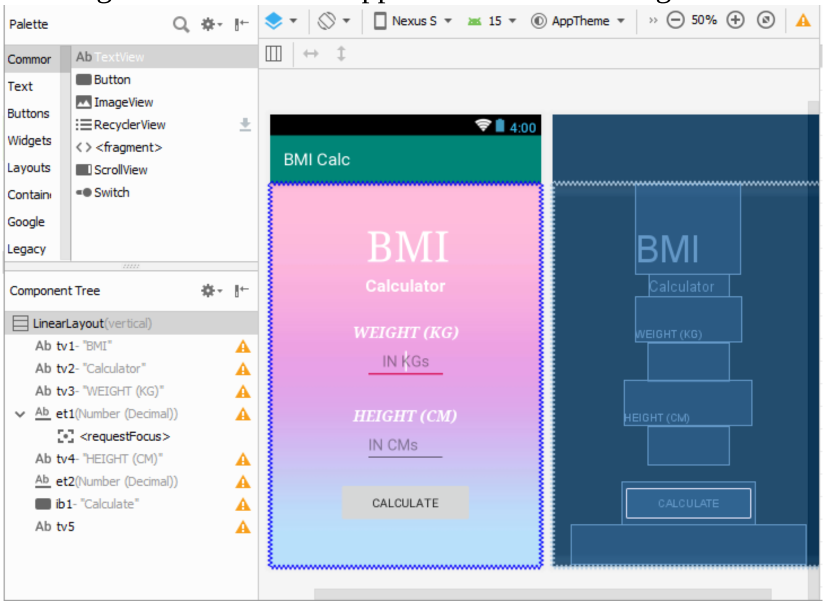
Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** shown below.
- Then delete the code which is there and type the code as given below.

```
<!-- Linear layout start here -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@drawable/images1"
  android:fadingEdge="horizontal"
  android:orientation="vertical" >

  <!-- Text view for BMI Text -->
  <TextView
    android:id="@+id/tv1"
    android:layout_width="124dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:paddingLeft="15dp"
    android:paddingTop="40dp"
    android:shadowColor="@android:color/black"
    android:shadowDx="4"
    android:shadowDy="4"
    android:text="BMI"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="@android:color/white"
    android:textSize="50sp"
    android:typeface="serif" />

  <!-- Textview for calculator text -->
  <TextView
    android:id="@+id/tv2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Calculator"
    android:textColor="@android:color/white"
    android:textSize="20dp"
    android:textStyle="bold" />

  <!-- Textview for WEIGHT(KG) text -->
  <TextView
    android:id="@+id/tv3"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:paddingTop="30dp"
        android:text="WEIGHT (KG)"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="@android:color/white"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <!-- Edit text for entering weight with hint in kgs -->
    <EditText
        android:id="@+id/et1"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:hint="IN KGs"
        android:ems="10"
        android:fadingEdgeLength="10dp"
        android:inputType="numberDecimal"
        android:textAlignment="center" >
        <requestFocus />
    </EditText>

    <!-- Text view for HEIGHT(CM)text -->
    <TextView
        android:id="@+id/tv4"
        android:layout_width="151dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:foregroundGravity="center_horizontal"
        android:gravity="center_horizontal"
        android:paddingTop="30dp"
        android:text="HEIGHT (CM)"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="@android:color/white"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <!-- Edit text for entering height with hint in cm -->
    <EditText
        android:id="@+id/et2"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:hint="IN CMs"
        android:ems="10"
        android:inputType="numberDecimal" >
    </EditText>

<!--Button for calculating the formula, when pressed, with calculate written over it-->
    <Button
        android:id="@+id/ib1"
        android:layout_width="158dp"
        android:layout_height="51dp"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:fadingEdge="vertical"
        android:longClickable="true"
        android:nextFocusRight="@android:color/holo_orange_dark"
        android:text="Calculate"
        android:visibility="visible" />

    <!-- Text view for showing result -->
    <TextView
        android:id="@+id/tv5"
        android:layout_width="278dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:gravity="center"
        android:paddingTop="20dp"
        android:text=""
        android:textColor="@android:color/holo_orange_dark"
        android:textSize="20dp"
        android:textStyle="bold" />

</LinearLayout>
   <!-- Linear layout ends here -->
```

Now click on the Design tab and now the application will look as given below.



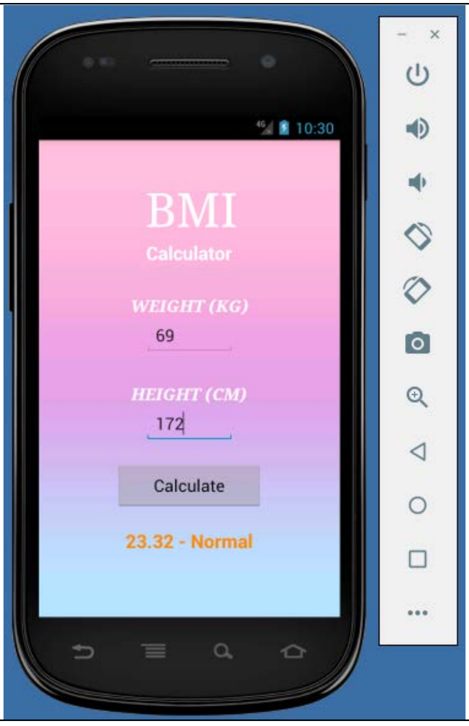
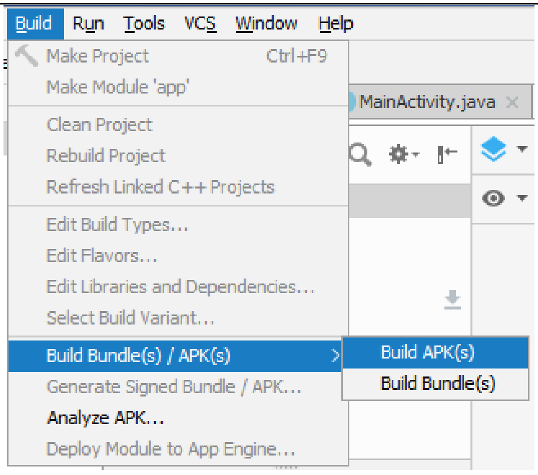
Java Coding for the Android Application:

- Click on **app -> java -> com.example. calcapp -> MainActivity.**
- Then delete the code which is there and type the code as given below.

```
package akn.bmicalc;

//Import necessary package and file
import android.os.Bundle;
import android.app.Activity;
import android.text.TextUtils;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

//Main activity class start here
public class MainActivity extends Activity {

   //Define layout
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

// Get the references to the widgets
```

```
        final EditText e1 = (EditText) findViewById(R.id.et1);
        final EditText e2 = (EditText) findViewById(R.id.et2);
        final TextView tv5 = (TextView) findViewById(R.id.tv5);

        findViewById(R.id.ib1).setOnClickListener(new View.OnClickListener() {

            // Logic for validation, input can't be empty
            @Override
            public void onClick(View v) {

                String str1 = e1.getText().toString();
                String str2 = e2.getText().toString();

                if(TextUtils.isEmpty(str1)){
                    e1.setError("Please enter your weight");
                    e1.requestFocus();
                    return;
                }

                if(TextUtils.isEmpty(str2)){
                    e2.setError("Please enter your height");
                    e2.requestFocus();
                    return;
                }

//Get the user values from the widget reference
                float weight = Float.parseFloat(str1);
                float height = Float.parseFloat(str2)/100;

//Calculate BMI value
                float bmiValue = calculateBMI(weight, height);

//Define the meaning of the bmi value
                String bmiInterpretation = interpretBMI(bmiValue);

                tv5.setText(String.valueOf(bmiValue + " - " + bmiInterpretation));

            }
        });

    }

    //Calculate BMI
    private float calculateBMI (float weight, float height) {
        return (float) Math.round((weight / (height * height))*100)/100;
    }

    // Interpret what BMI means
    private String interpretBMI(float bmiValue) {

        if (bmiValue < 16) {
            return "Severely underweight";
        } else if (bmiValue < 18.5) {

            return "Underweight";
        } else if (bmiValue < 25) {

            return "Normal";
        } else if (bmiValue < 30) {

            return "Overweight";
        } else {
            return "Obese";
        }
    }
}
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below.	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
 BMI Calculator; WEIGHT (KG) 69; HEIGHT (CM) 172; Calculate; 23.32 - Normal	 Build > Build Bundle(s) / APK(s) > Build APK(s)

Result:

Thus a Simple Android Application to design a Simple BMI Calculator is developed and executed successfully in emulator and Mobile device.

Ex. No. 11.
Date:

Displaying images using Multithreading

Aim:

To develop a Simple Android Application to display the images using multithreading

Procedure:

Creating a New project:

- Open Android Studio and then click on **File -> New -> New project.**
- Then type the Application name as "**Multithread"**, and click **Next.**
- Then select the **Minimum SDK** as shown below and click **Next**.
- Then select the **Empty Activity** and click **Next.**
- Finally click F**inish**.

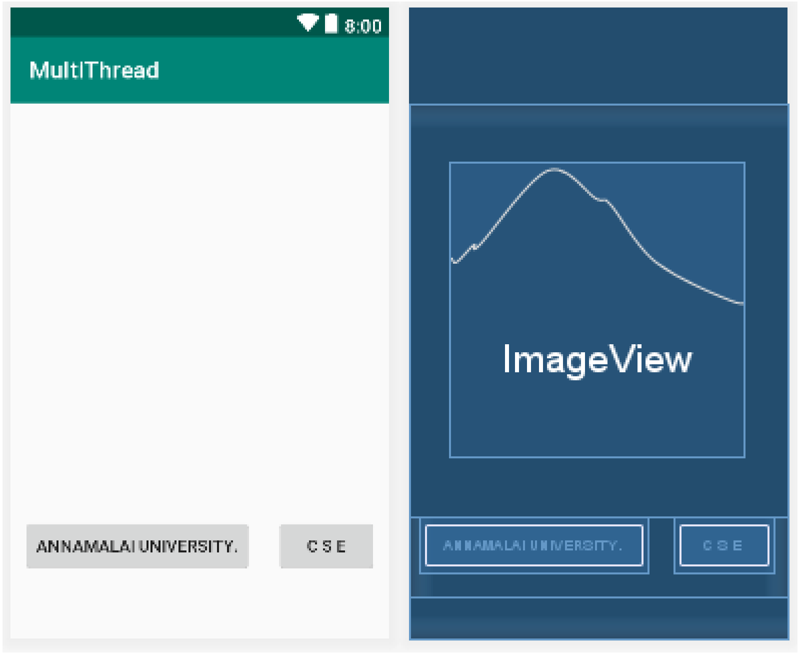
Designing layout for the Android Application:

- Click on **app -> res -> layout -> activity_main.xml.**
- Now click on **Text** shown below.
- Then delete the code which is there and type the code as given below.

```
<?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" >

  <ImageView
    android:id="@+id/imageView"
    android:layout_width="250dp"
    android:layout_height="250dp"
    android:layout_gravity="center"
    android:layout_margin="50dp" />

  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="horizontal">

    <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="left"
      android:layout_margin="10dp"
      android:text="Annamalai University." />

    <Button
      android:id="@+id/button2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:layout_gravity="right"
        android:layout_margin="10dp"
        android:text=" C S E " />
    </LinearLayout>
</LinearLayout>
```

Now click on the Design tab and now the application will look as given below.



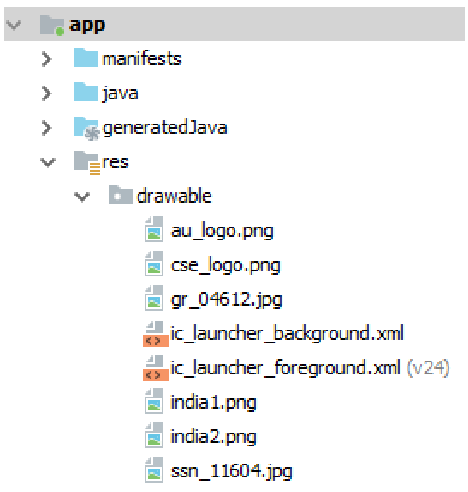
Java Coding for the Android Application:

Java Coidng for Main Activity:

- Click on **app -> java -> com.example.multithread -> MainActivity.**
- Then delete the code which is there and type the code as given below.

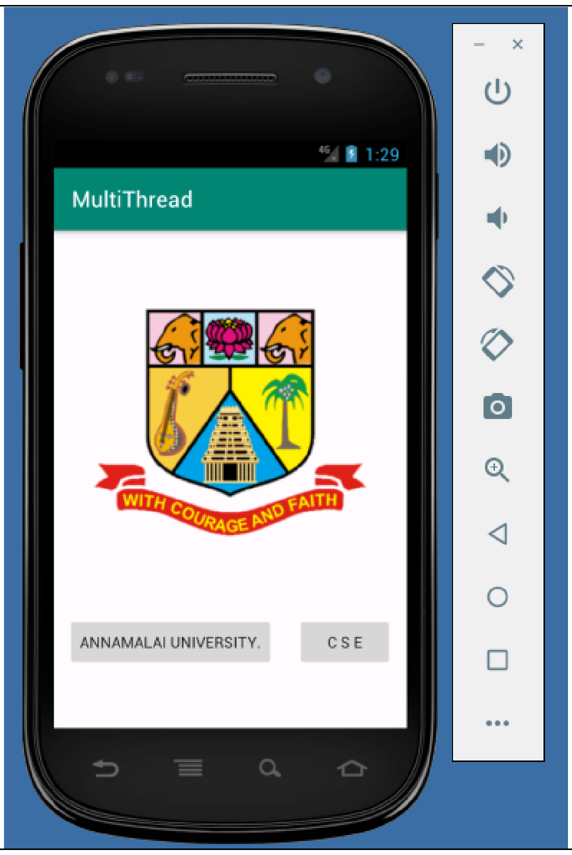
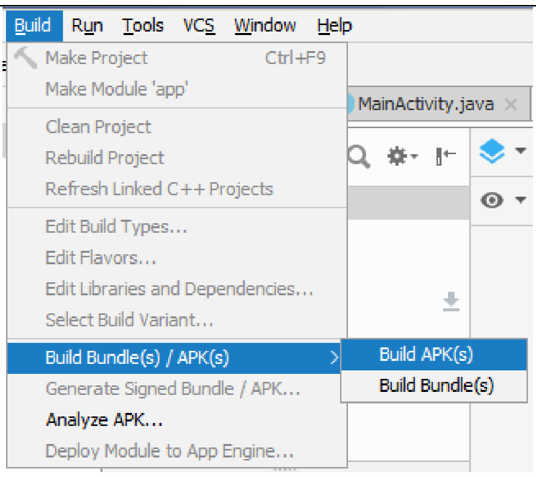
```
package com.example.cse.multithread;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
public class MainActivity extends AppCompatActivity
{
    ImageView img;
    Button bt1,bt2;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        bt1 = (Button)findViewById(R.id.button);
        bt2= (Button) findViewById(R.id.button2);
        img = (ImageView)findViewById(R.id.imageView);

        bt1.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                new Thread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        img.post(new Runnable()
                        {
                            @Override
                            public void run()
                            {
                                img.setImageResource(R.drawable.au_logo);
                            }
                        });
                    }
                }).start();
            }
        });

        bt2.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                new Thread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        img.post(new Runnable()
                        {
                            @Override
                            public void run()
                            {
                                img.setImageResource(R.drawable.cse_logo);
                            }
                        });
                    }
                }).start();
            }
        });
    }
}
```

- Copy the images to be displayed and locate the drawable under app->res->drawable and paste it under ic_launcher_background.xml as shown.
- Image should
    - be png type
    - have the file size < 100kb
    - not have the resolution of > 300 x 300
    - have the file name using only the characters from a-z, 0-9 and underscore



```
app
  manifests
  java
  generatedJava
  res
    drawable
      au_logo.png
      cse_logo.png
      gr_04612.jpg
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
      india1.png
      india2.png
      ssn_11604.jpg
```

Click Play icon or press Shift+F10 and select any of the Available Virtual devices from the list or create a New AVD.

Emulator will be loaded and displays the output of the app developed as shown below. (If the analog clock is not supported, then the digital clock will be displayed.)	Select Build APK(s) from Build Menu and build the .apk file for this application. Locate the apk file created and copy it to the mobile Phone to install it and run it from it for verification.
 1:29 MultiThread WITH COURAGE AND FAITH ANNAMALAI UNIVERSITY. CSE	 Build Run Tools VCS Window Help Make Project Ctrl+F9 Make Module 'app' Clean Project Rebuild Project Refresh Linked C++ Projects Edit Build Types... Edit Flavors... Edit Libraries and Dependencies... Select Build Variant... Build Bundle(s) / APK(s) Generate Signed Bundle / APK... Analyze APK... Deploy Module to App Engine... Build APK(s) Build Bundle(s) MainActivity.java

Result:

Thus a Simple Android Application to display the image using multithreading is developed and executed successfully in emulator and Mobile device.