Mobile Application Development: Lab 1

Objectives

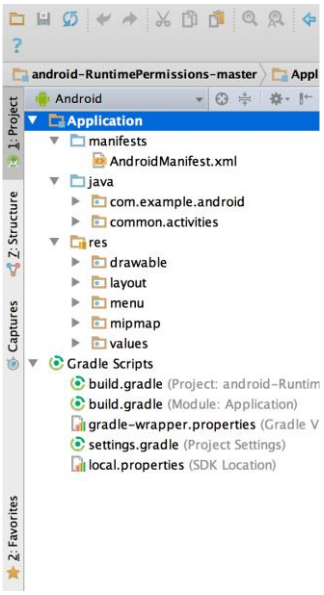
- Understanding Android Studio
- Understanding Android Templates
- Practice Activities

Objective 1: Understanding Android Studio

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

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

Project Structure:



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

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

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

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

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

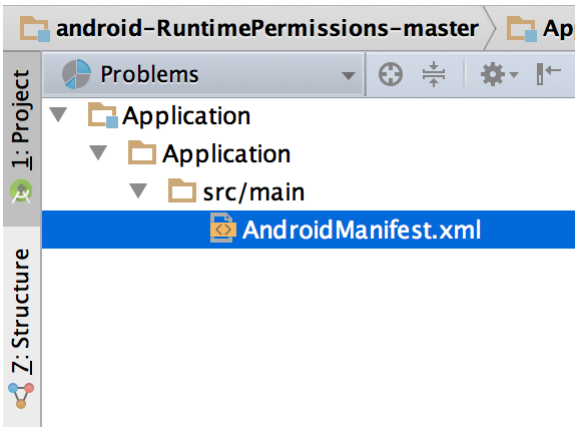


Figure 2. The project files in Problems view, showing a layout file with a problem

Android Studio Lookout:

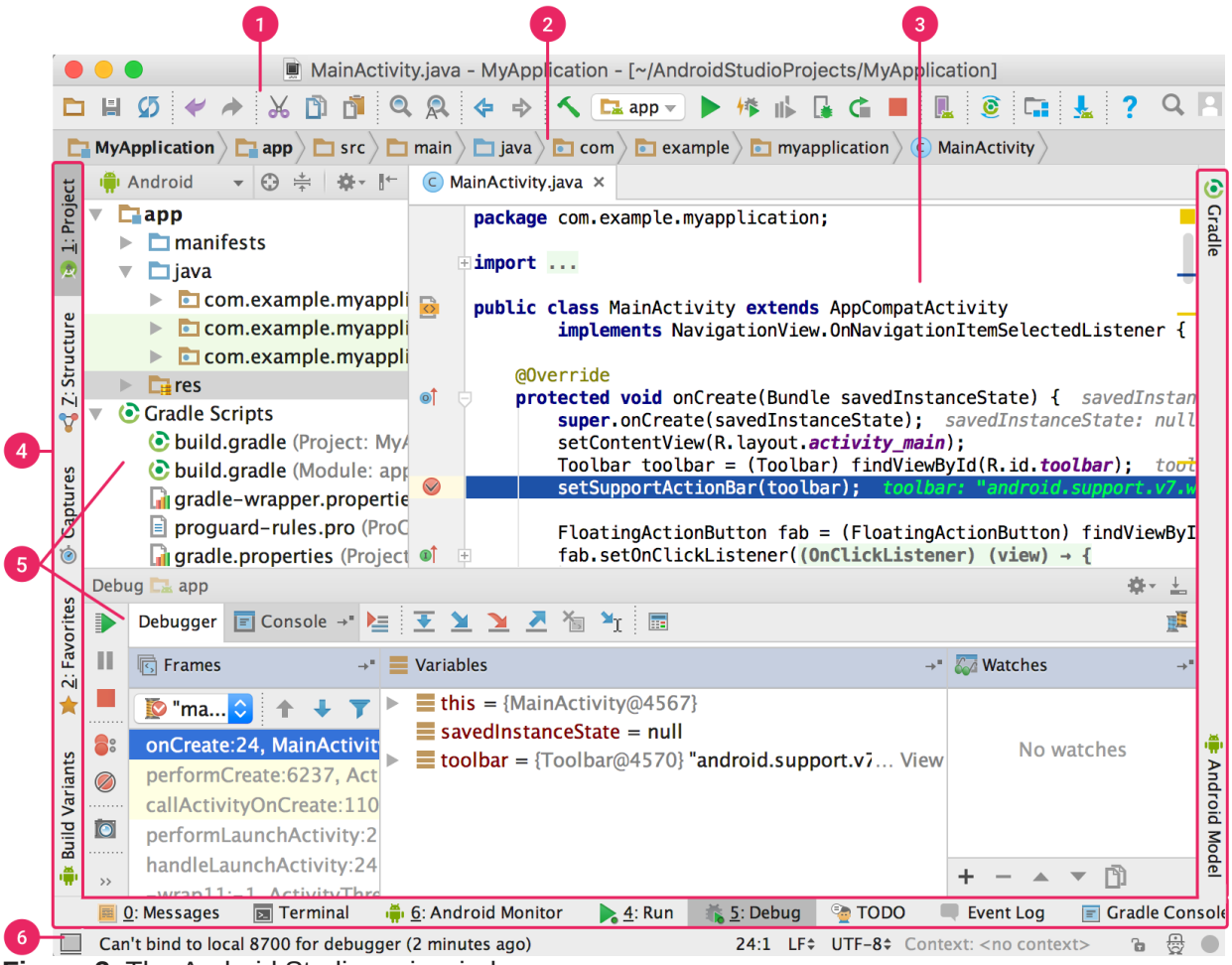


Figure 3. The Android Studio main window.

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

Gradle build system

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

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

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

Build variants

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

Multiple APK support

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

Resource shrinking

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

Debug and profile tools

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

Inline debugging

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

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

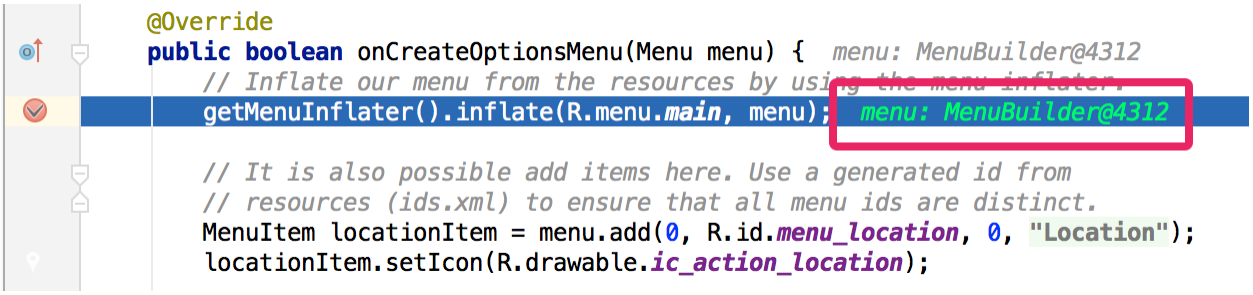


Figure 4: *An inline variable value.*

Performance profilers

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

Heap dump

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

Memory Profiler

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

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 you compile your program, Android Studio automatically runs configured **Lint** and other **IDE inspections** to help you easily identify and correct problems with the structural quality of your code.

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

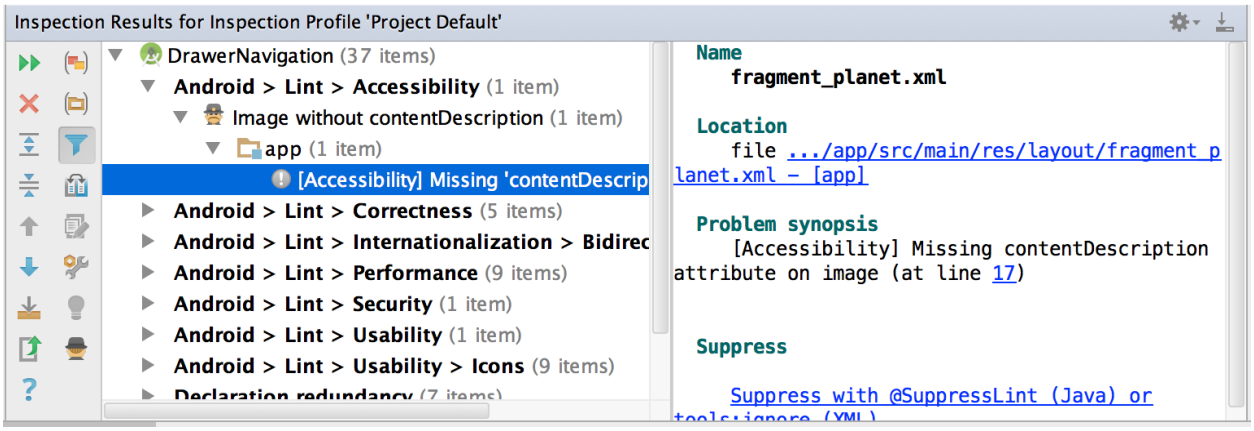


Figure 5: *The results of a Lint inspection in Android Studio.*

The Android project view

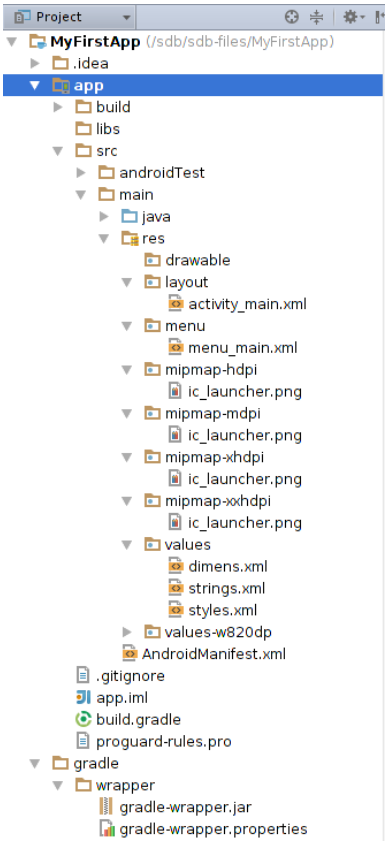
To see the actual file structure of the project including all files hidden from the Android view, select **Project** from the dropdown at the top of the **Project** window.

When you select **Project** view, you can see a lot more files and directories. The most important of which are the following:

module-name/

- build/

  Contains build outputs.



- libs/

Contains private libraries.

- src/

Contains all code and resource files for the module in the following subdirectories:

  - androidTest/

Contains code for instrumentation tests that run on an Android device. For more information, see the Android Test documentation.

  - main/

Contains the "main" sourceset files: the Android code and resources shared by all build variants (files for other build variants reside in sibling directories, such as src/debug/ for the debug build type).

    - AndroidManifest.xml

Describes the nature of the application and each of its components. For more information, see the AndroidManifest.xml documentation.

    - java/

Contains Java code sources.

    - jni/

Contains native code using the Java Native Interface (JNI). For more information, see the Android NDK documentation.

    - gen/

Contains the Java files generated by Android Studio, such as your R.java file and interfaces created from AIDL files.

    - res/

Contains application resources, such as drawable files, layout files, and UI string. See Application Resources for more information.

    - assets/

Contains file that should be compiled into an .apk file as-is. You can navigate this directory in the same way as a typical file system using URIs and read files as a stream of bytes using the AssetManager . For example, this is a good location for textures and game data.

  - test/

Contains code for local tests that run on your host JVM.

- build.gradle **(module)**

This defines the module-specific build configurations.

- build.gradle **(project)**

This defines your build configuration that apply to all modules. This file is integral to the project, so you should maintain them in revision control with all other source code.

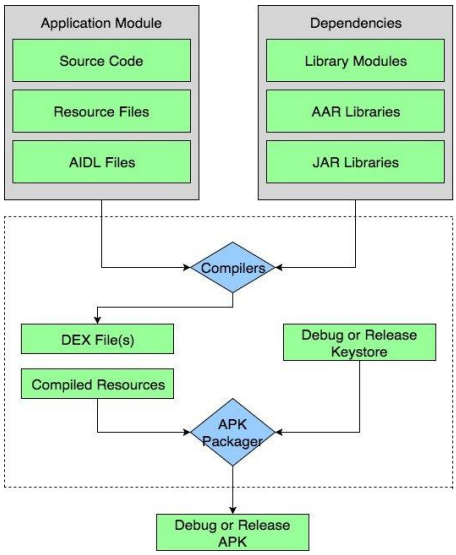
Gradle

Gradle is a build system, which is responsible for code compilation, testing, deployment and conversion of the code into .dex files and hence running the app on the device.

As Android Studio comes with Gradle system pre-installed, there is no need to install additional runtime softwares to build our project. Whenever you click on **Run** button in android studio, a gradle task automatically triggers and starts building the project and after gradle completes its task, app starts running in AVD or in the connected device.

A build system like Gradle is not a compiler, linker etc, but it controls and supervises the operation of compilation, linking of files, running test cases, and eventually bundling the code into an apk file for your Android Application.

There are two build.gradle files for every android studio project of which, one is for **application** and other is for **project level(module level) build files**.



In the build process, the compiler takes the source code, resources, external libraries JAR files and AndroidManifest.xml(which contains the meta-data about the application) and convert them into .dex(Dalvik Executable files) files, which includes bytecode. That bytecode is supported by all android devices to run your app. Then **APK Manager** combines the .dex files and all other resources into single **apk** file. **APK Packager** signs debug or release apk using respective debug or release keystore.

Debug apk is generally used for testing purpose or we can say that it is used at development stage only. When your app is complete with desired features and you are ready to publish your application for external use then you require a **Release apk** signed using a release keystore.

Gradle Build Process

You can even start your gradle system through **command line tool**. Following commands are used for it:

- ./gradlew build - (build project)
- ./gradlew clean build - (build project complete scratch)
- ./gradlew clean build - (run the test)
- ./gradlew wrapper - (to see all the available tasks)

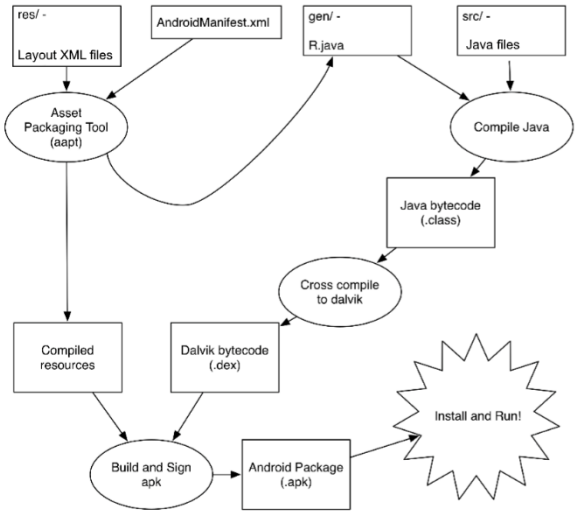


Figure 1: Gradle Build Process

ART - Android Runtime

The **Dalvik Virtual Machine is dead**. Yes, Google stopped using it in 2014, although you will find most of the Android tutorials online, still not updated, but please be informed that Dalvik Virtual Machine is not used in Android anymore.

The new runtime is known as ART or Android Runtime which is very well compatible with its predecessor Dalvik, but do comes in with a lot of new features like:

- **Ahead-of-Time compilation**
- **Improved Garbage collection**
- Improved Debugging and diagnostics.

Objective 2: Understanding Android Live Templates

- Live Templates are pre-built code chunks/ lines of code used for rapid application development.
- One can use already created templates or can also create their own templates.

⇨ Inserting Live templates

Inserting live templates is the easiest thing in android, android always does this by itself, whenever you write the code, just pressing the tab will automatically insert the related chunk of code.

- ➢ To Browse the current Live Templates:
- ➢ Go to **File > Settings > Editor > Live Templates**.

⇨ Make you own Live Template

- Just write the code, highlight it, and go to Tools > Save as Live Template.

Code completion

Table 2. Keyboard shortcuts for code completion.

Type	Description	Windows and Linux	Mac
Basic Completion	Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.	Control+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

Objective 3: Practice Activities

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

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

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

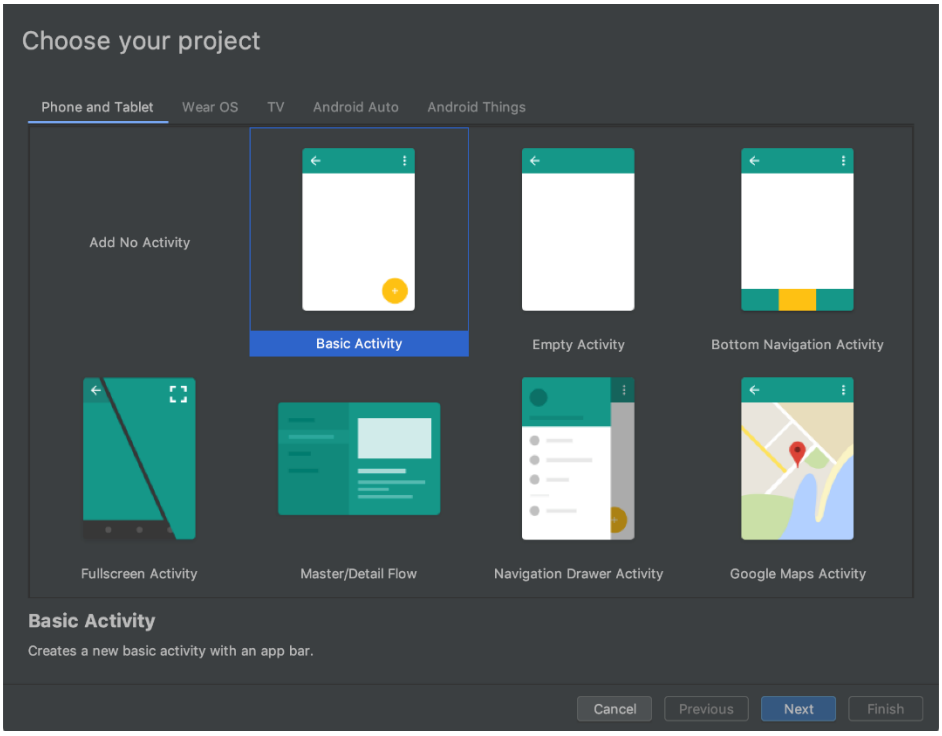


Figure 2: In the first screen of the wizard, choose the type of project you want to create.

- **Configure your project**
    - The next step is to configure some settings and create your new project, as described below and shown in figure 2.

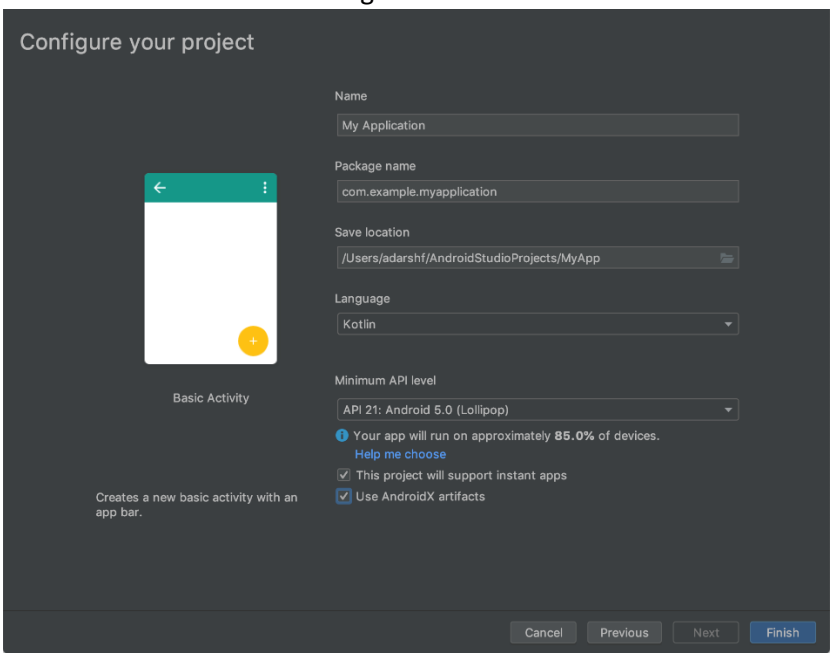


Figure 3: Configure your new project with a few settings.

1. Specify the **Name** of your project.
2. Specify the **Package name**. By default, this package name also becomes your application ID, which you can change later.
3. Specify the **Save location** where you want to locally store your project.
4. Select the **Language** you want Android Studio to use when creating sample code for your new project. Keep in mind, you are not limited to using only that language creating the project.
5. Select the **Minimum API level** you want your app to support. When you select a lower API level, your app can rely on fewer modern Android APIs. However, a larger percentage of Android devices are able to run your app. The opposite is true when selecting a higher API level. If you want to see more data to help you decide, click **Help me choose**.
6. If you want your project to use AndroidX libraries by default, which are improved replacements of the Android Support libraries, check the box next to **Use AndroidX artifacts**. To learn more, read the AndroidX overview.
7. When you're ready to create your project, click **Finish**.

App Basics

Our first application will be a simple application with one activity and a layout, in which we will just print **Hello World!** on the device screen. Here are a few things that you must know, don't worry about it, you will understand all these, as we move forward:

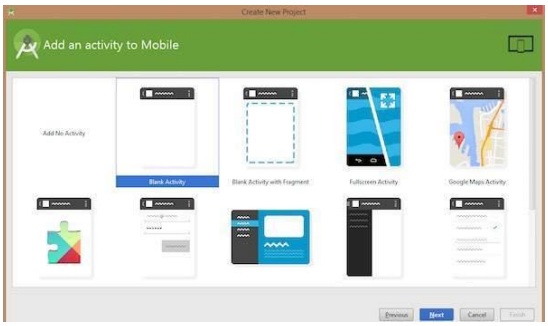
- An **activity** is an instance of Activity, a class in the Android SDK. An activity is responsible for managing user interaction with a screen of information. Whenever we create a new activity, we

are actually writing a subclass of Activity class. A simple application may need only one subclass, while a complex application can have many. In our first application FirstAppActivity will be our activity subclass.

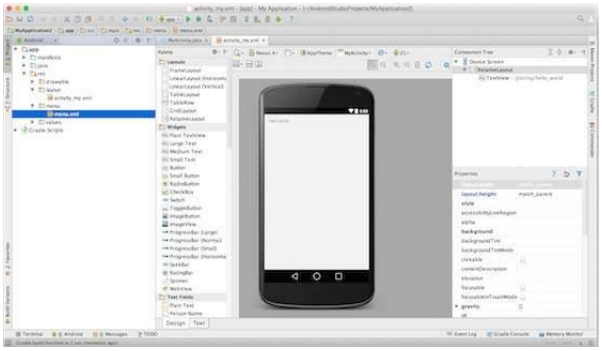
- A **layout** defines a set of user interface(UI) objects(Views and Widgets) and their position on the device screen. A layout is made up of definitions written in XML. Each definition is used to create an object(a user interface) that appears on the device's screen, like a button or some text or an image.

Let's build a sample project

- Create an empty screen project from menu as told above



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



- The Main Activity File

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

class MainActivity : AppCompatActivity() {

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

Mobile Application Development: Lab 1

- The Layout File

```
<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" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      android:text="@string/hello_world"
      tools:context=".MainActivity" />

</RelativeLayout>
```

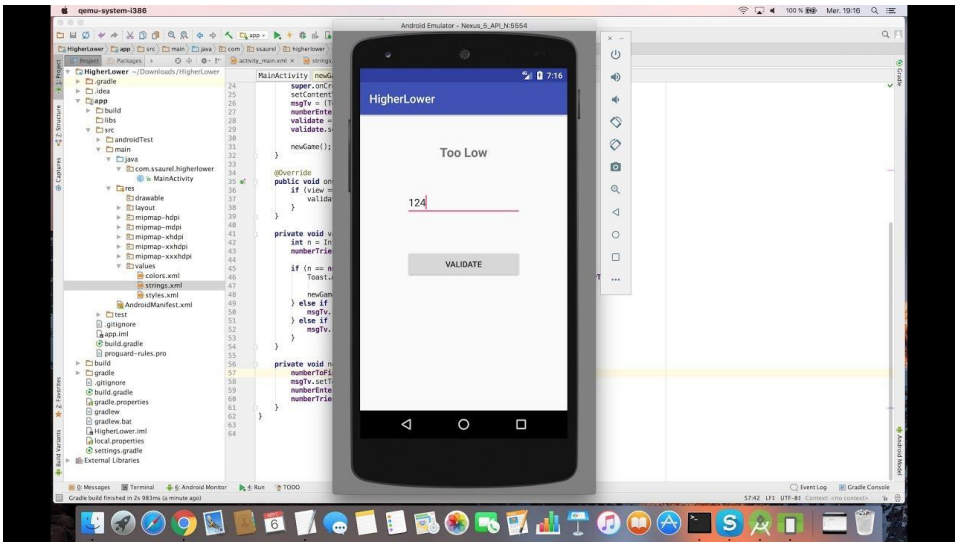
- The Strings File

```
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="hello_world">Hello world!</string>
   <string name="menu_settings">Settings</string>
   <string name="title_activity_main">MainActivity</string>
</resources>
```

Mobile Application Development: Lab 1

Actvitiy 1:

Random number guessing game**:** The computer thinks of a number from 1 to 1000. The user makes guesses, and after each incorrect guess, the app hints to the user whether the right answer is higher or lower than their guess. The game ends when the player guesses the right number.



The components to add

View	Value	Event
TextView	(Too Low, Too High or You Guessess Correctly)	None
EditText		
Button	Validate	Validate()

Hint: Following Code

```
Random random = Random()
var numToGuess = random.nextInt(1000)
```

```
fun validate (View view) {
    var inputField = findViewById<EditText>(R.id.textBoxId)
    var textLabel = findViewById<TextView>(R.id.textLabel)
    val number = Integer.parseInt(inputField.text.toString())

    if(number > numToGuess)
        textLabel.text = "Your guess is Too High"
    else if(number < numToGuess)
        textLabel.text = "Your guess is Too Low"
    else
        textLabel.text = "Hoorah!! ... You guessed it correctly"

}
```

Mobile Application Development: Lab 1

Activity 2:

Memory Game: Several buttons are shown on screen, "face down", and the user needs to try to find pairs that match up when flipped over.

Hint: You can use arrays to store the values of the images as Adjacency Array.

Download Icons Freely : https://www.flaticon.com/packs

https://www.iconfinder.com/search/?q=image

Activity 3:

Mr. Potato Head *(thanks to Victoria Kirst for original assignment idea and images!)* Write an app that displays a "Mr. Potato Head" toy on the screen as an `ImageView`. The toy has several accessories and body parts that can be placed on it, such as eyes, nose, mouth, ears, hat, shoes, and so on. We will provide you with image files for each body part and accessory, such as **body.png**, **ears.png**, **hat.png**, and so on. Initially your image view should display only the toy's body, but if the user checks/unchecks any of the check boxes below the toy, the corresponding body part or accessory should appear/disappear. The way to display the various body parts is to create a separate `ImageView` for each part, and lay them out in the XML so that they are superimposed on top of each other. You can achieve this with a `RelativeLayout` in which you give every image the same position, though you should probably nest it in some other overall layout for the screen. The check boxes should align themselves into a grid of rows and columns. You can set whether or not an image (or any other widget) is visible on the screen by setting its `android:visibility` property in the XML, and/or by calling its `setVisibility` method in your Java code. The `setVisibility` method accepts a parameter such as `View.VISIBLE` or `View.INVISIBLE`. There is also a `getVisibility` method if you need to check whether a widget is currently visible.