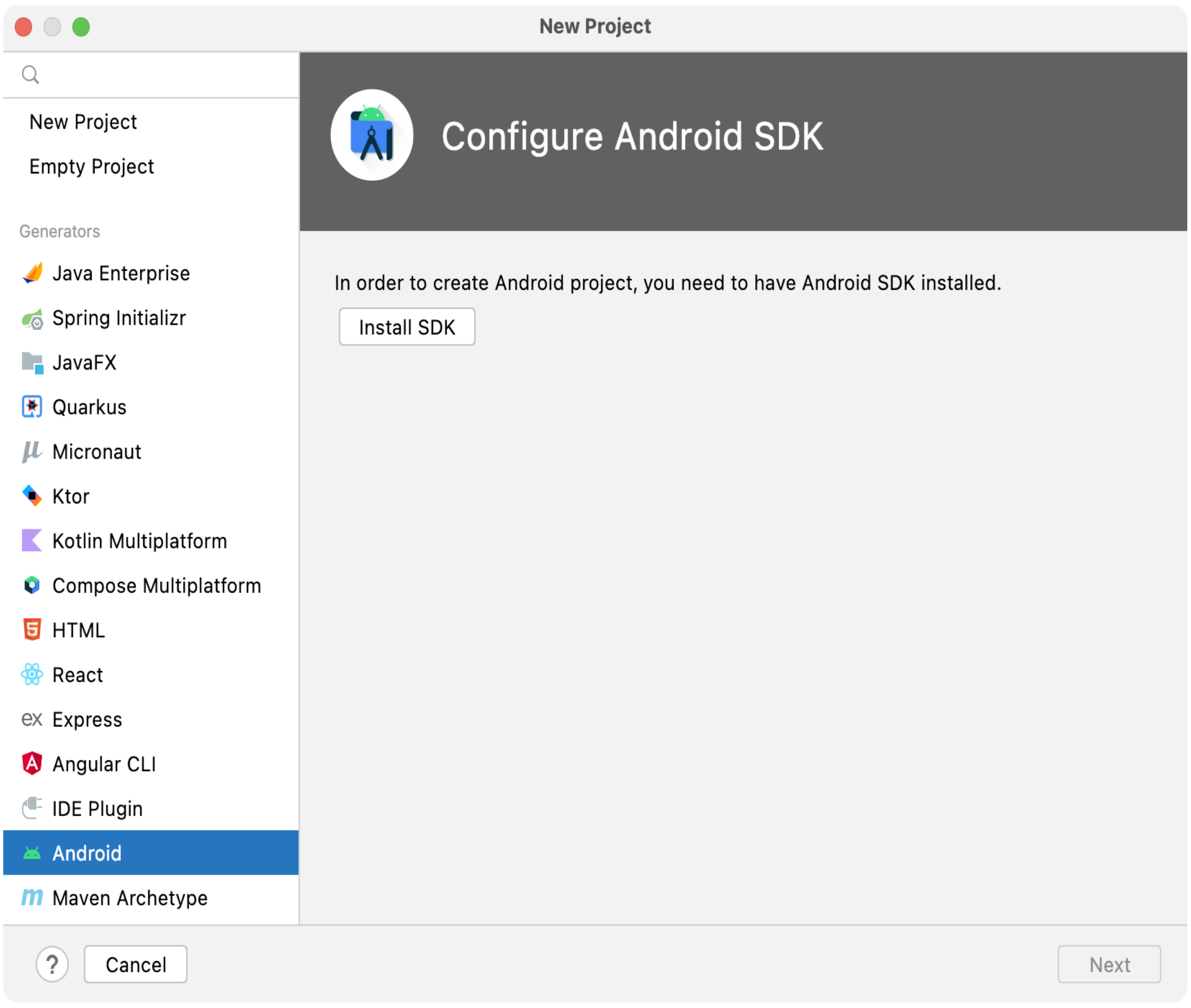
Create a new Android project

Create a project

1. Launch IntelliJ IDEA. On the Welcome screen, click New Project. If you already have a project open, from the main menu select File | New | Project.
2. In the New Project wizard, select Android on the left.

   If you don't have the Android SDK configured, IntelliJ IDEA will detect this and prompt you to download it:



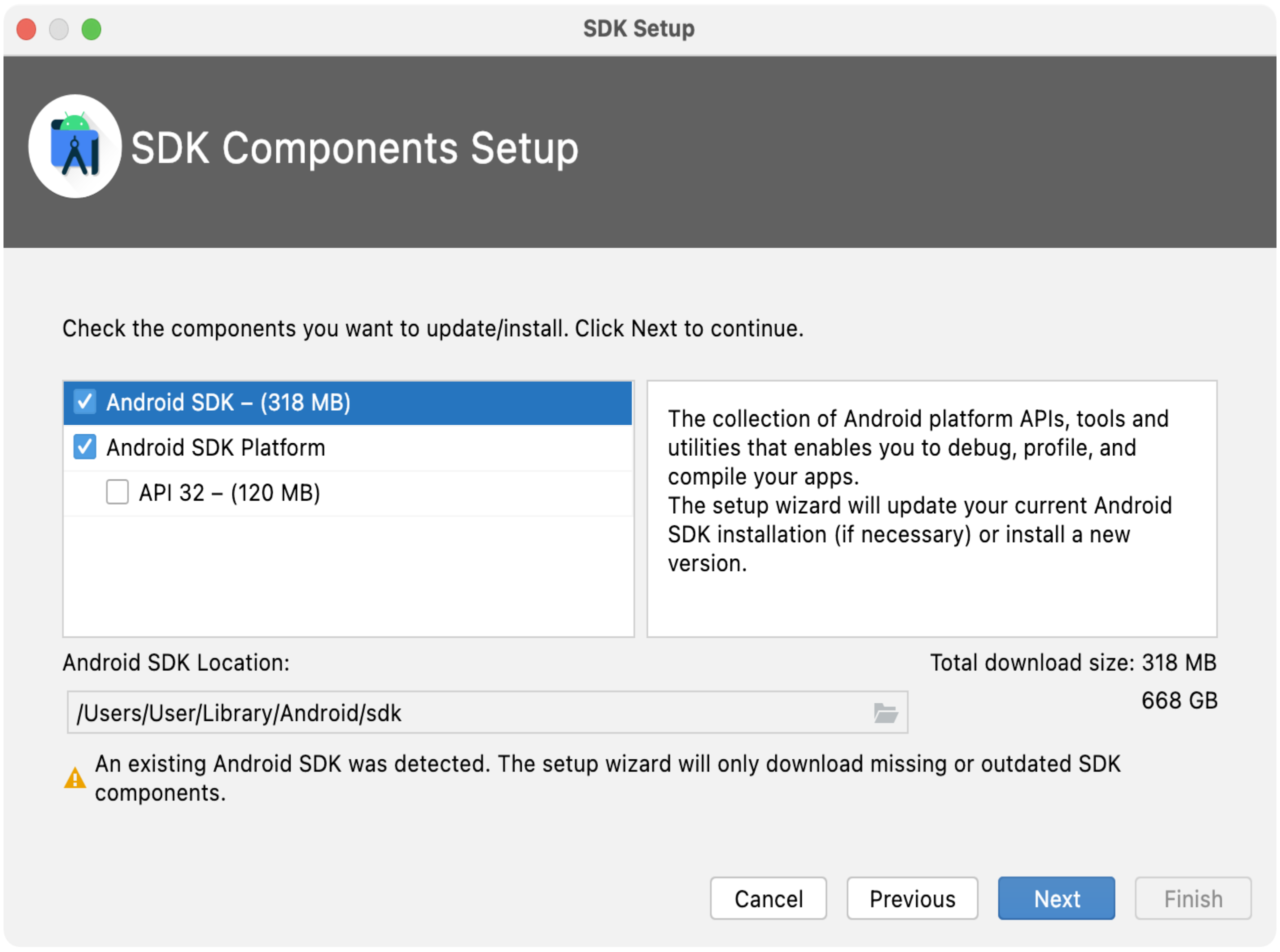
New Project

Configure Android SDK

In order to create Android project, you need to have Android SDK installed.

3. Select the components you want to install. If you haven't installed the Android SDK tools before, all the required components will be preselected.

   Optionally, modify the location for the Android SDK, and click Next:



Check the components you want to update/install. Click Next to continue.

- [✓] Android SDK – (318 MB)
- [✓] Android SDK Platform
  - [ ] API 32 – (120 MB)

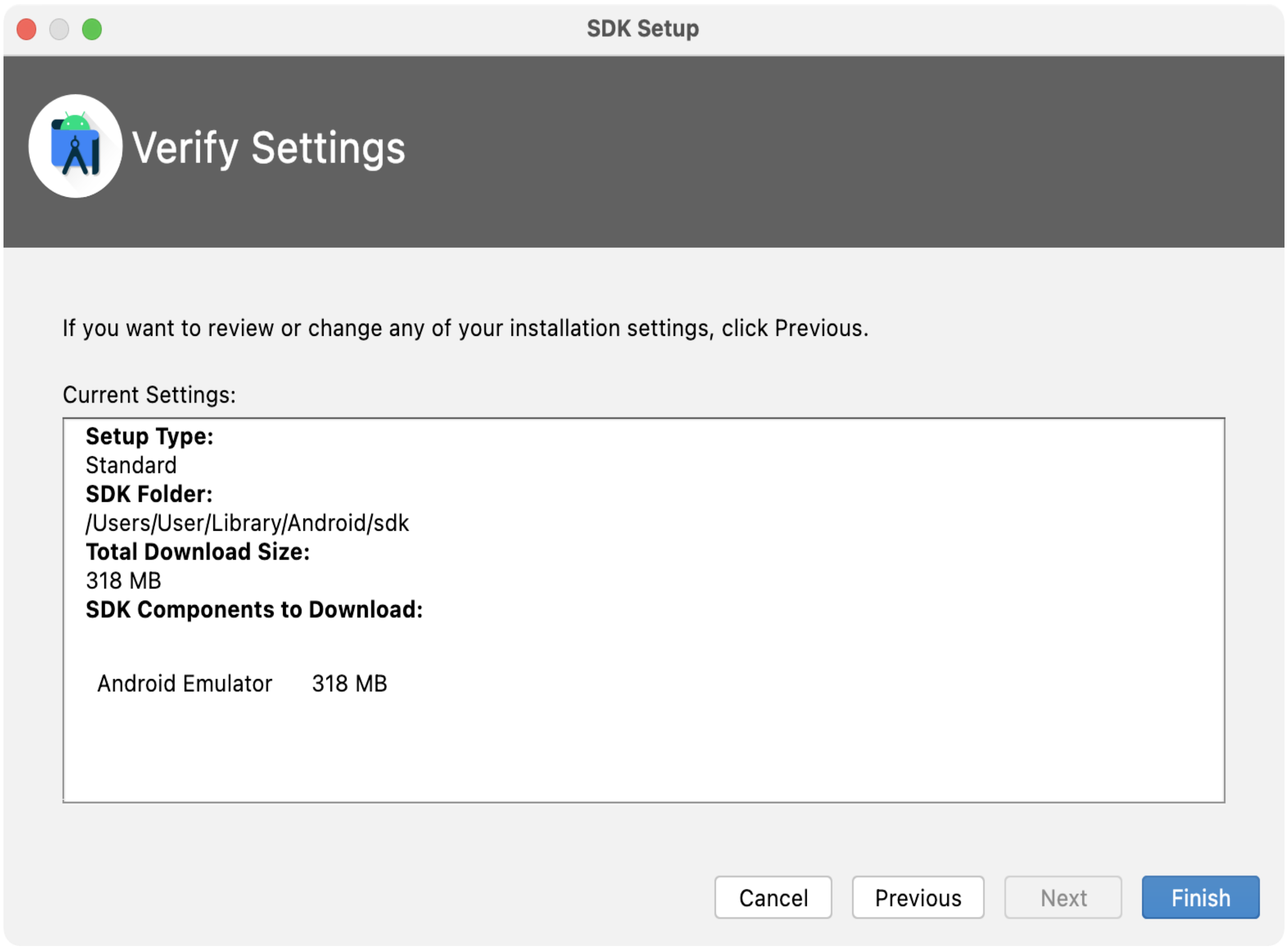
The collection of Android platform APIs, tools and utilities that enables you to debug, profile, and compile your apps.
The setup wizard will update your current Android SDK installation (if necessary) or install a new version.

Android SDK Location: /Users/User/Library/Android/sdk

Total download size: 318 MB
668 GB

An existing Android SDK was detected. The setup wizard will only download missing or outdated SDK components.

4. Review the installation settings and click Finish to start the download:



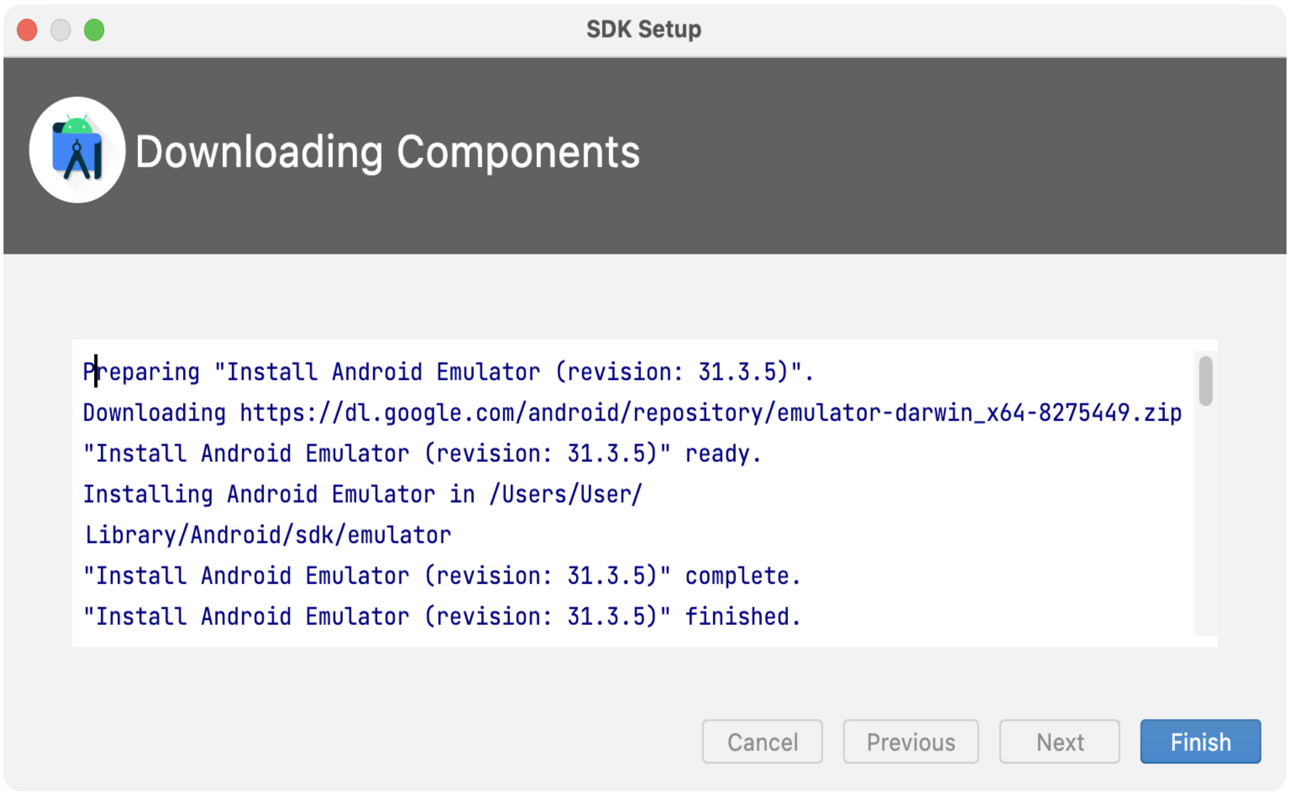
If you want to review or change any of your installation settings, click Previous.

Current Settings:

Setup Type:
Standard
SDK Folder:
/Users/User/Library/Android/sdk
Total Download Size:
318 MB
SDK Components to Download:

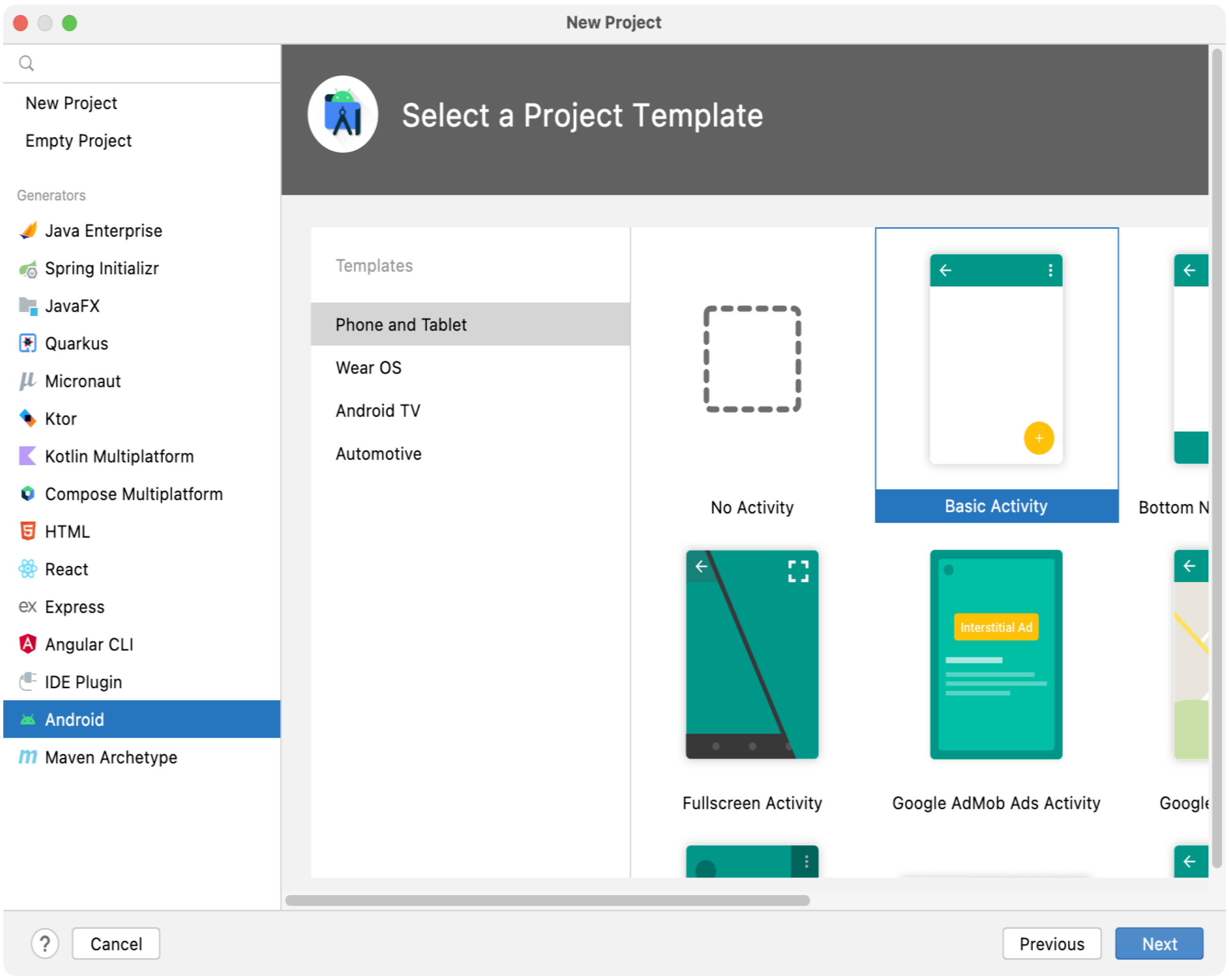
Android Emulator	318 MB

5. When all components have been downloaded and installed, click Finish:

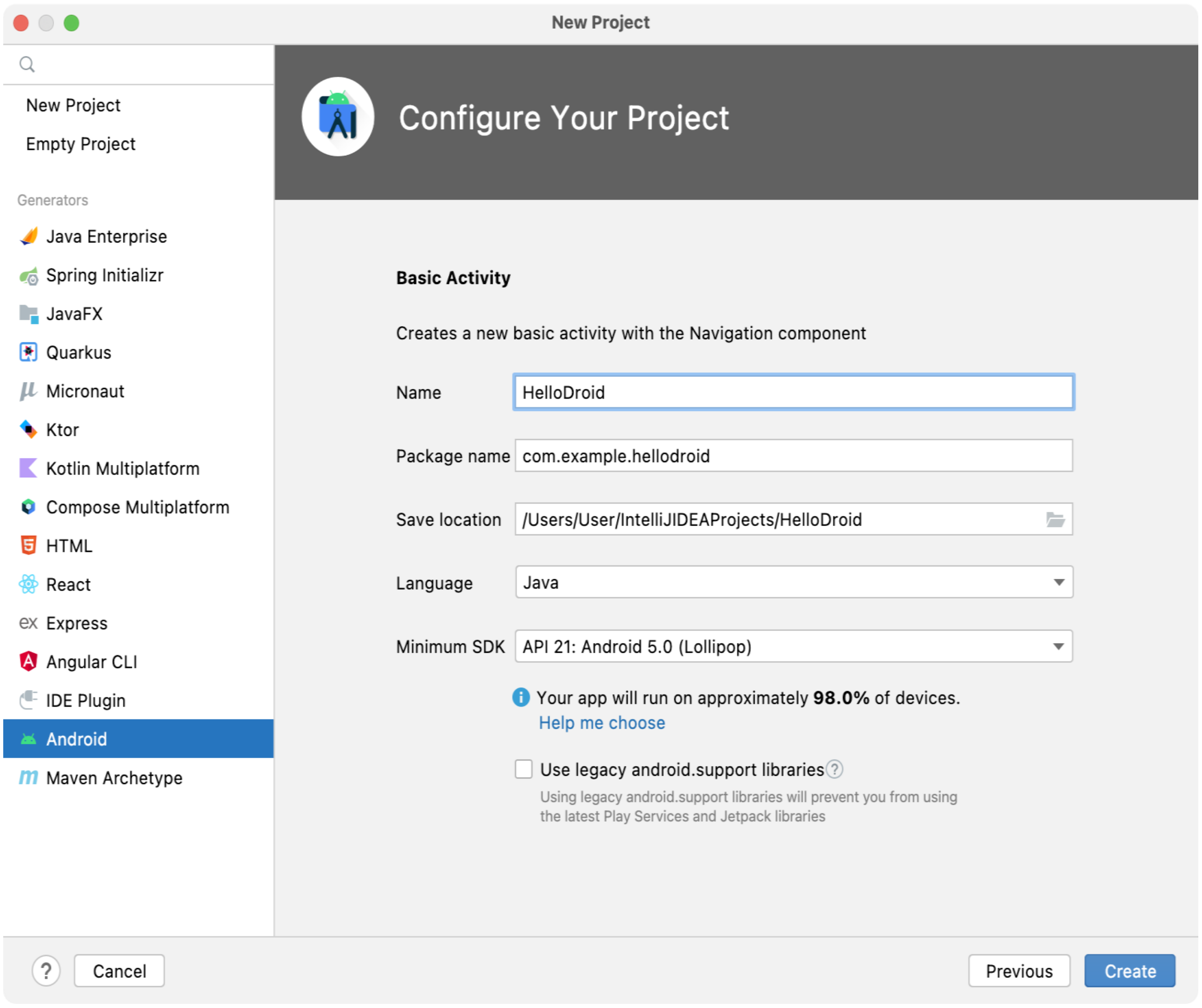


```
Preparing "Install Android Emulator (revision: 31.3.5)".
Downloading https://dl.google.com/android/repository/emulator-darwin_x64-8275449.zip
"Install Android Emulator (revision: 31.3.5)" ready.
Installing Android Emulator in /Users/User/
Library/Android/sdk/emulator
"Install Android Emulator (revision: 31.3.5)" complete.
"Install Android Emulator (revision: 31.3.5)" finished.
```

6. Select Empty Activity as the project template:



7. On the last step, type `HelloDroid` as the project name and select Java as the language:



Basic Activity

Creates a new basic activity with the Navigation component

Name: HelloDroid

Package name: com.example.hellodroid

Save location: /Users/User/IntelliJIDEAProjects/HelloDroid

Language: Java

Minimum SDK: API 21: Android 5.0 (Lollipop)

Your app will run on approximately **98.0%** of devices.
Help me choose

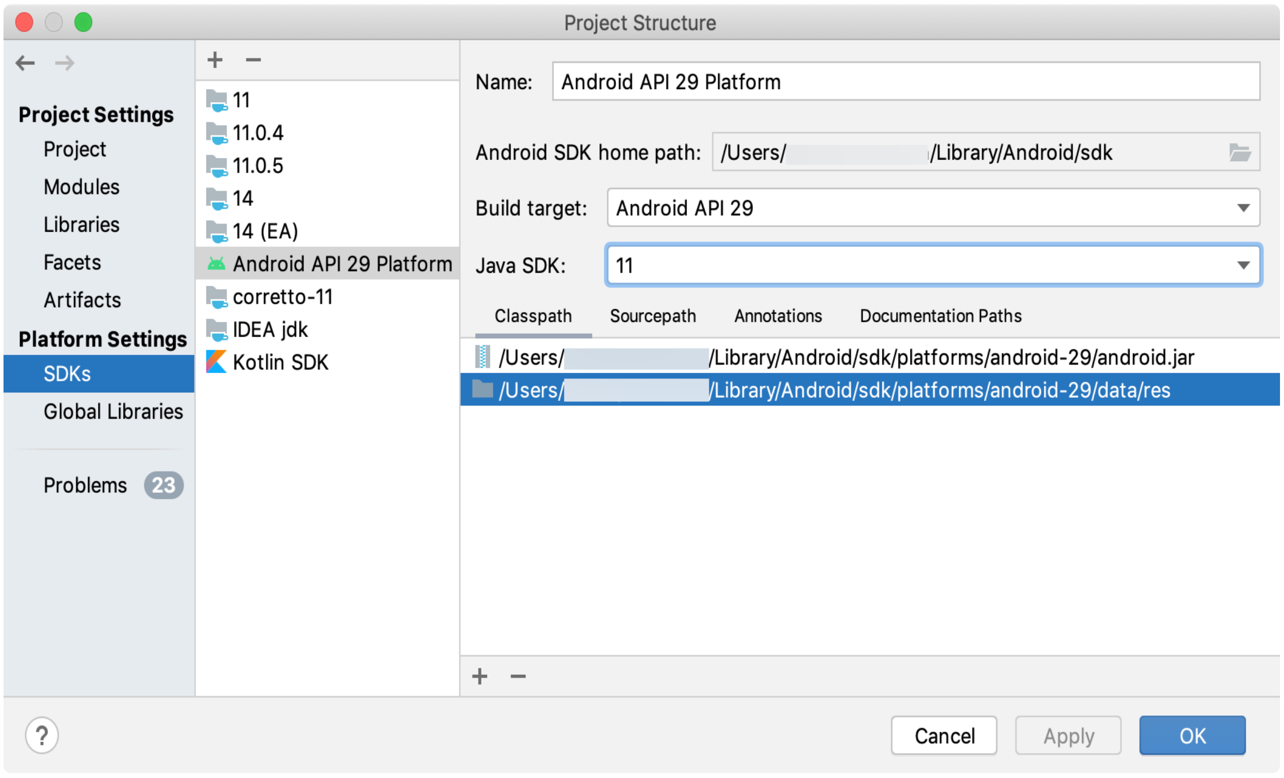
☐ Use legacy android.support libraries

Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

Configure project JDK

Now that we have created our first project, let's make sure it uses the correct JDK.

1. Go to File | Project Structure and go to Platform Settings | SDKs. Select the Android SDK and make sure that the correct Java version is selected in the Java SDK field.


Project Structure

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems 23

11
11.0.4
11.0.5
14
14 (EA)
Android API 29 Platform
corretto-11
IDEA jdk
Kotlin SDK

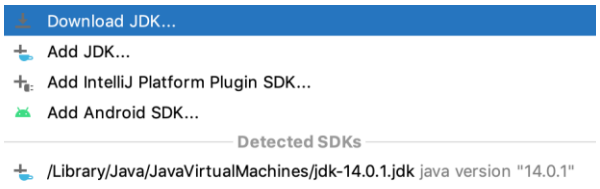
Name: Android API 29 Platform
Android SDK home path: /Users/[redacted]/Library/Android/sdk
Build target: Android API 29
Java SDK: 11

Classpath | Sourcepath | Annotations | Documentation Paths

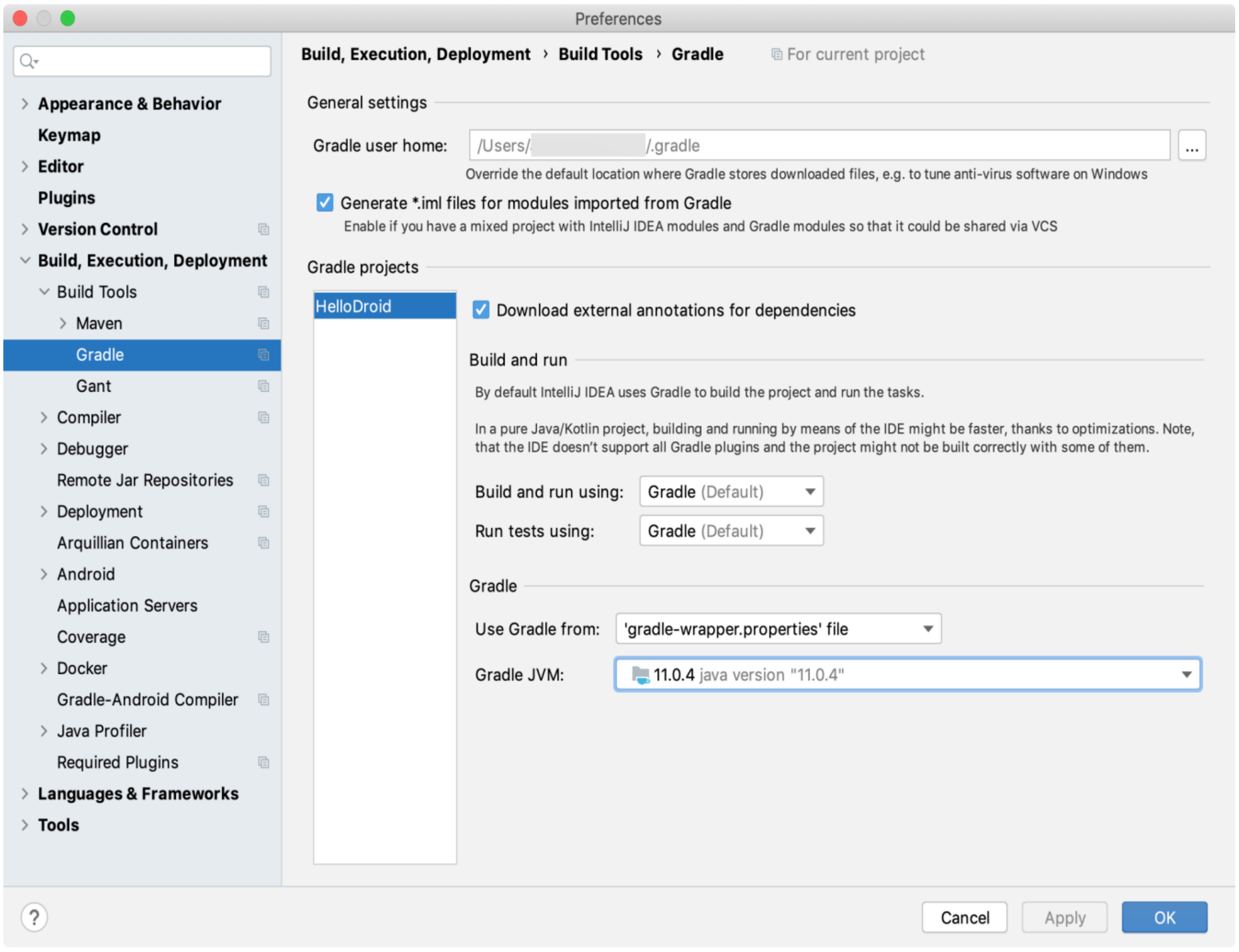
/Users/[redacted]/Library/Android/sdk/platforms/android-29/android.jar
/Users/[redacted]/Library/Android/sdk/platforms/android-29/data/res

Cancel | Apply | OK

We recommend that you use Java SE 11 or Java SE 8 for Android development in IntelliJ IDEA. If you don't have the correct JDK installed, in the Project Structure dialog, click the Add New SDK button + on the toolbar and select Download JDK:


Download JDK...
Add JDK...
Add IntelliJ Platform Plugin SDK...
Add Android SDK...
Detected SDKs
/Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk java version "14.0.1"

2. In the Settings dialog (CtrlAlt0S), go to Build, Execution, Deployment | Build Tools | Gradle and select the correct Java version (8.x or 11.x).


Preferences

Build, Execution, Deployment › Build Tools › Gradle   For current project

General settings

Gradle user home: /Users/[redacted]/.gradle

Override the default location where Gradle stores downloaded files, e.g. to tune anti-virus software on Windows

☑ Generate *.iml files for modules imported from Gradle

Enable if you have a mixed project with IntelliJ IDEA modules and Gradle modules so that it could be shared via VCS

Gradle projects

HelloDroid

☑ Download external annotations for dependencies

Build and run

By default IntelliJ IDEA uses Gradle to build the project and run the tasks.

In a pure Java/Kotlin project, building and running by means of the IDE might be faster, thanks to optimizations. Note, that the IDE doesn't support all Gradle plugins and the project might not be built correctly with some of them.

Build and run using: Gradle (Default)

Run tests using: Gradle (Default)

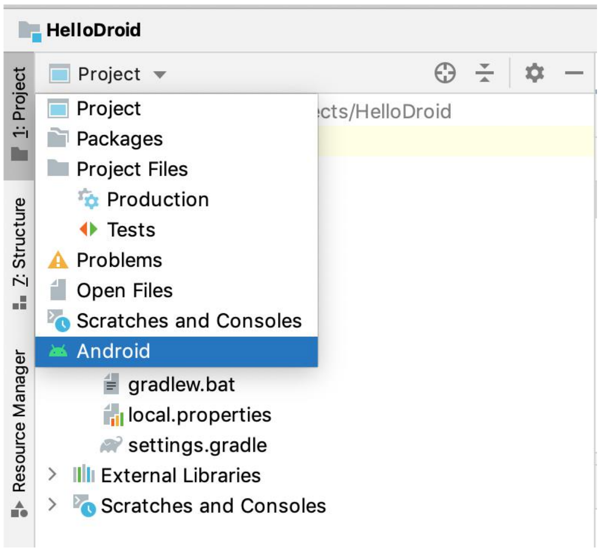
Gradle

Use Gradle from: 'gradle-wrapper.properties' file

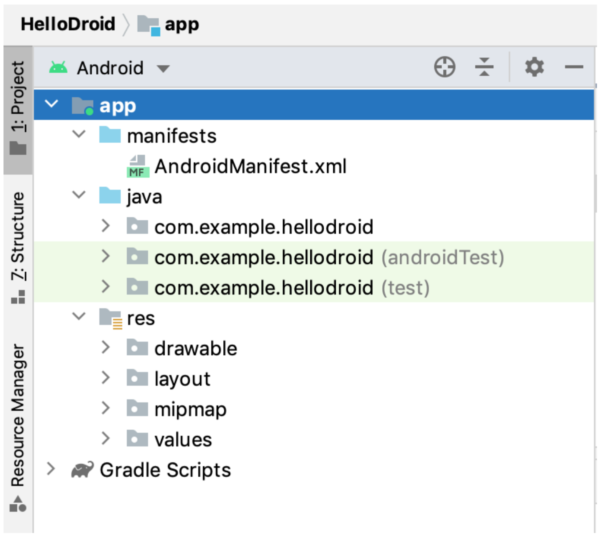
Gradle JVM: 11.0.4 java version "11.0.4"

Explore the project structure

For Android projects, there's a dedicated view in the IntelliJ IDEA Project tool window: click Project in the top-left corner and select Android.



This view doesn't reflect the actual hierarchy of files on your disk – it is organized by modules and file types to ease navigation between source files of your project. Note that it hides project files and directories that you don't commonly use (to see them, choose the Project view):



```
app
  manifests
    AndroidManifest.xml
  java
    com.example.hellodroid
    com.example.hellodroid (androidTest)
    com.example.hellodroid (test)
  res
    drawable
    layout
    mipmap
    values
Gradle Scripts
```

The app folder consists of the following subfolders:

- manifests: contains the AndroidManifest.xml file, which holds general information about the application processed by the Android operating system. Among other things, it declares the package name that serves as a unique identifier for your application and the minimum version of the Android SDK required for the device where the application will run. It also declares the entry points of the application, along with permissions the application requires. For more information, refer to App Manifest Overview.
- java: contains the Java source code files grouped by packages, including JUnit tests.
- res: contains all non-code resources, such as XML layout files, UI strings, images, and so on.

The Gradle Scripts folder contains all the project's build-related configuration files.

Edit the UI layout

At this stage, the user interface of our sample `HelloDroid` application is based on a very simple layout defined in the `activity_main.xml` file located in the `res/layout` folder.

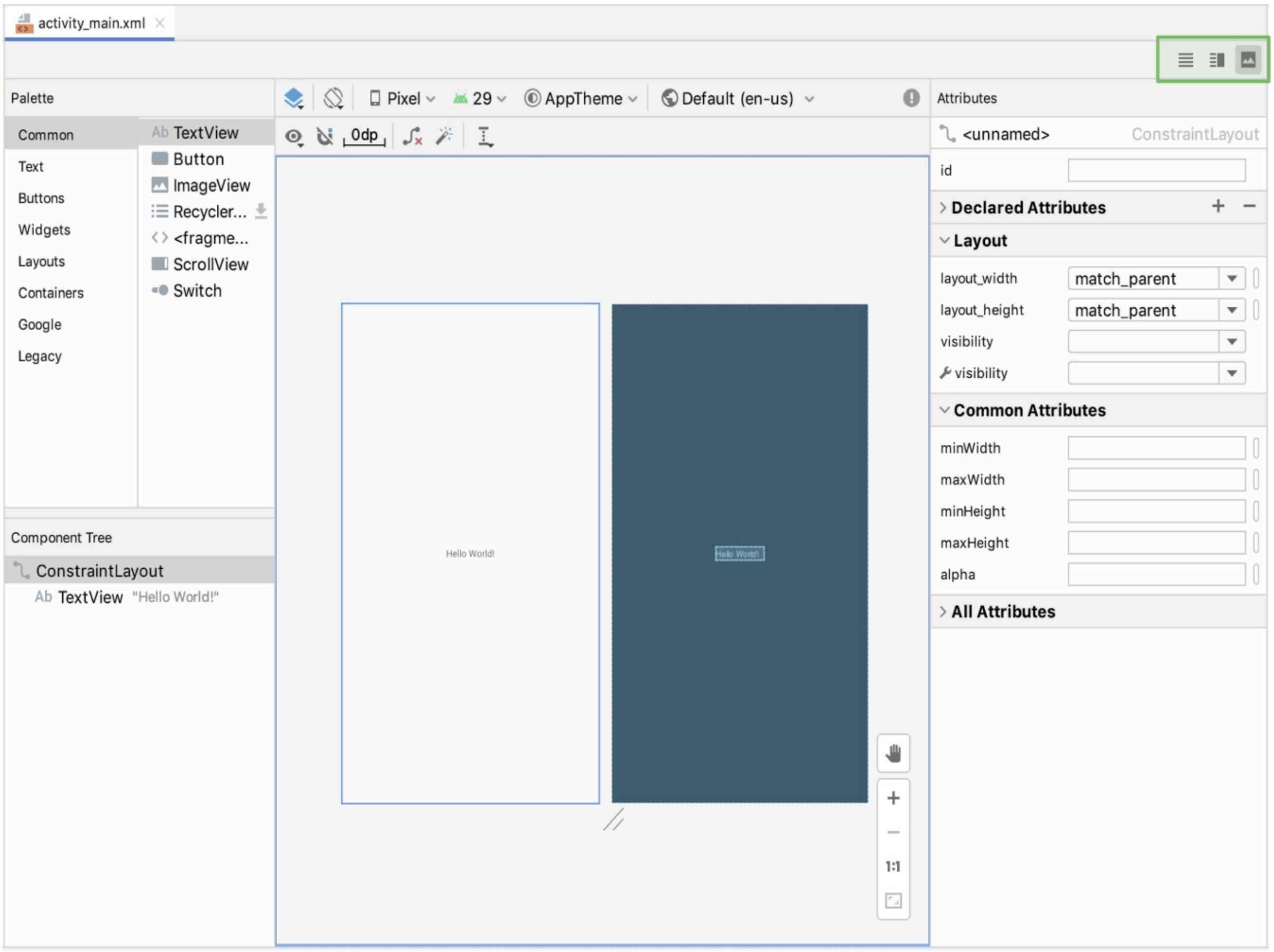
let's modify the auto-generated user interface and see how the application layout is rendered without running it on any physical or virtual device.

Open the UI designer

1. In the Android project view, go to the `app/res/layout` and double-click the `activity_main.xml` file to open it. Note that since IntelliJ IDEA downloads the components required to render layout files, opening it may take a few seconds.

If the UI designer fails to open, and you get the Design editor is unavailable until after a successful project sync error, press CtrlShift0A, search for the `Sync Project with Gradle Files` action, and wait for the sync to finish.

By default, IntelliJ IDEA provides a graphical view of the layout file, but you can also switch to the source code view, or view the text and the graphical representation side by side – use the icons in the top-right corner of the UI Designer pane:

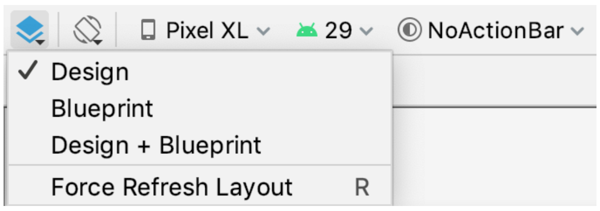


layout_width: match_parent
layout_height: match_parent
visibility
visibility

This pane shows a rectangular canvas that is synchronized with the layout definition and with the Component Tree, so any changes to the canvas are reflected there accordingly.

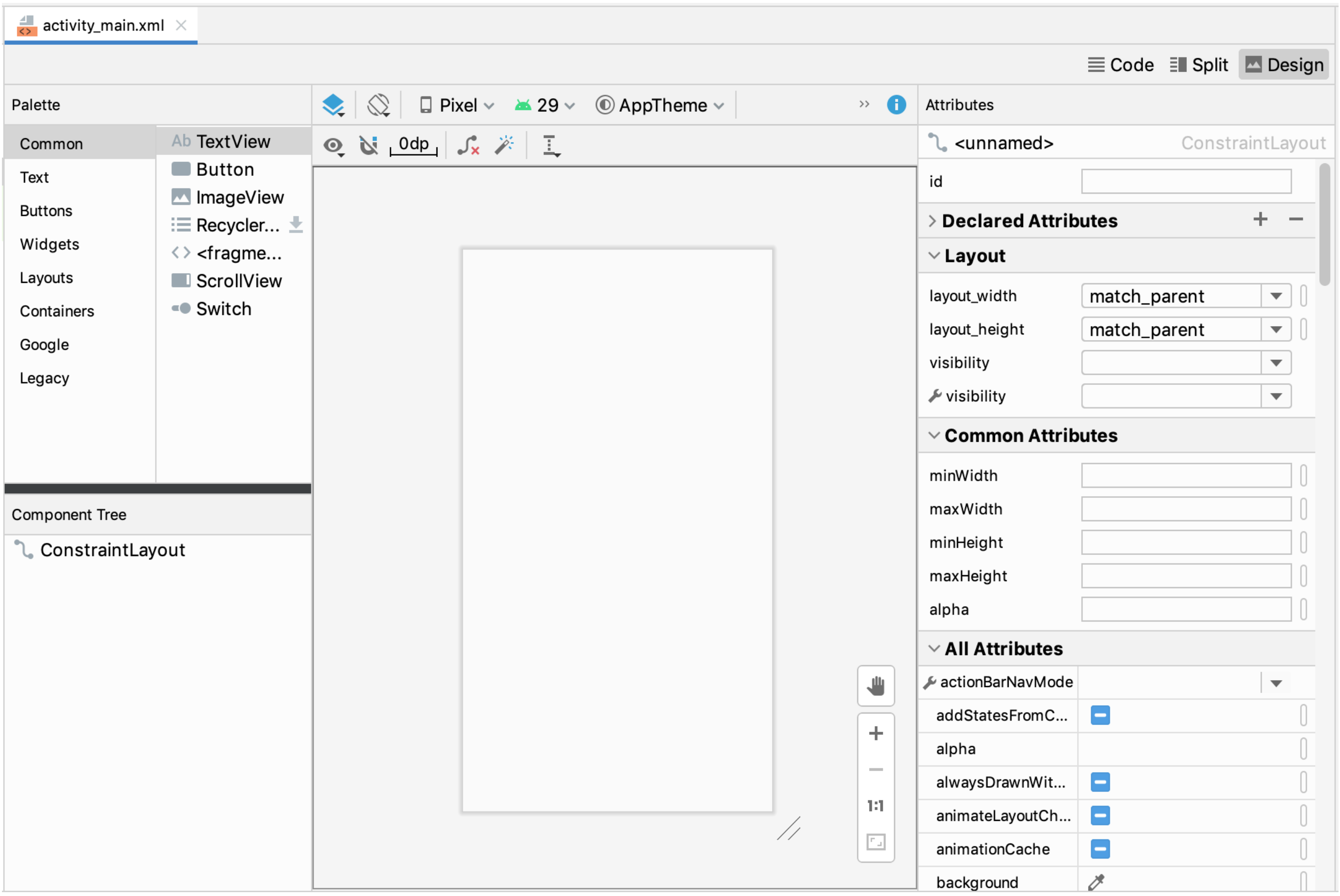
Normally, layout files have a layout manager as their root element (for example, `LinearLayout`, `FrameLayout`, `ConstraintLayout`, and so on). In our example, the root element in activity_main.xml is `ConstraintLayout` that is responsible for positioning the elements of the application interface. For the purpose of this tutorial, we are not going to modify it, but you can learn more about designing interfaces from Build a Responsive UI with ConstraintLayout.

2. To eliminate distraction and only see how your layout is represented, click the Select Design Surface icon in the top-left corner and choose Design:



3. Now let's delete the existing text element. To do this, right-click the text label and choose Delete from the context menu.

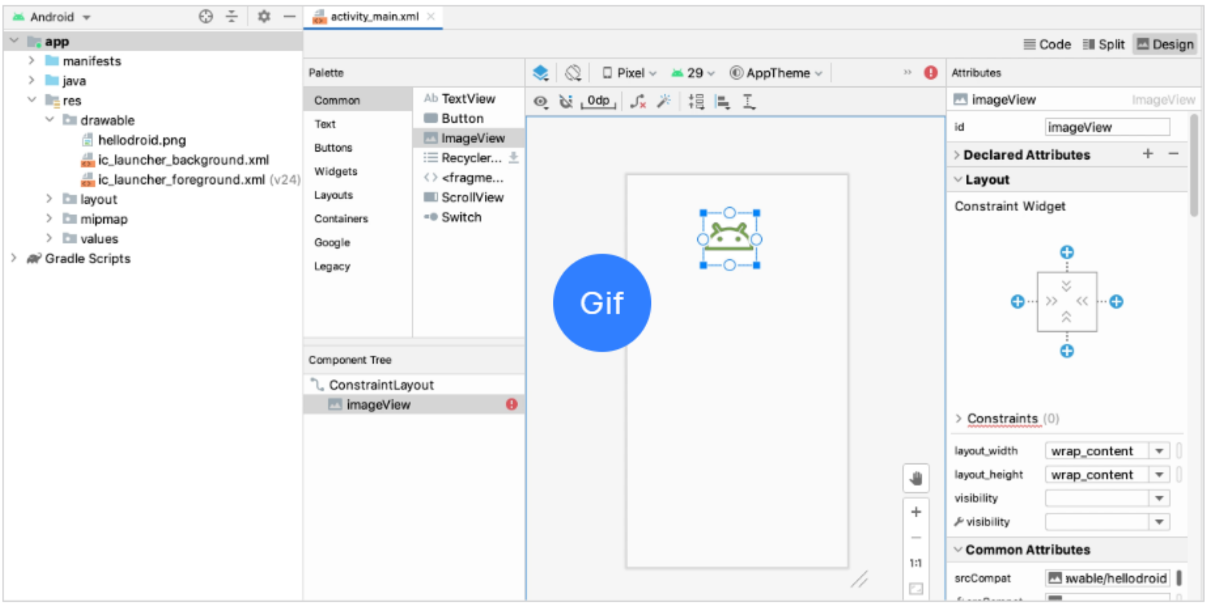
Now the UI layout looks like the following, and we are ready to start designing the layout of our application:



Add image to the UI layout

Now let's add a droid image to our layout.

1. In the Android project view, expand the `app/res` folder and drag the image you want to use into the `drawable` folder. For this tutorial, we've downloaded a Hello Droid image from the Internet and saved it with the dimensions 50x50 px.
2. Return to the `activity_main.xml` file opened in the Designer pane, from the Palette choose the ImageView element, and drag it to the canvas to the position where you want the image to appear.
3. In the Pick a Resource dialog that opens, choose the resource file you've added and click OK:



id: imageView

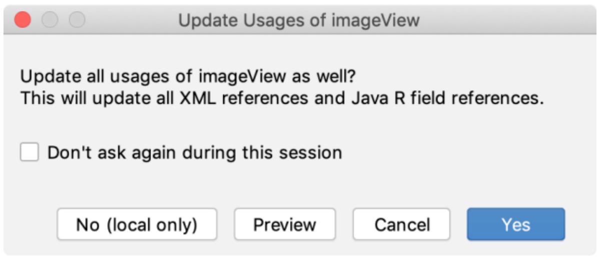
layout_width: wrap_content

layout_height: wrap_content

srcCompat: awable/hellodroid

4. Next, we need to modify the default id of the `imageView` element to be able to reference it later.

   Select it in the Component Tree and in the Attributes pane on the right, enter the new identifier in the id field: `droidImage`. Press Enter; in the dialog that opens, confirm that you want to update all references to the image element id:



Update Usages of imageView

Update all usages of imageView as well?
This will update all XML references and Java R field references.

☐ Don't ask again during this session

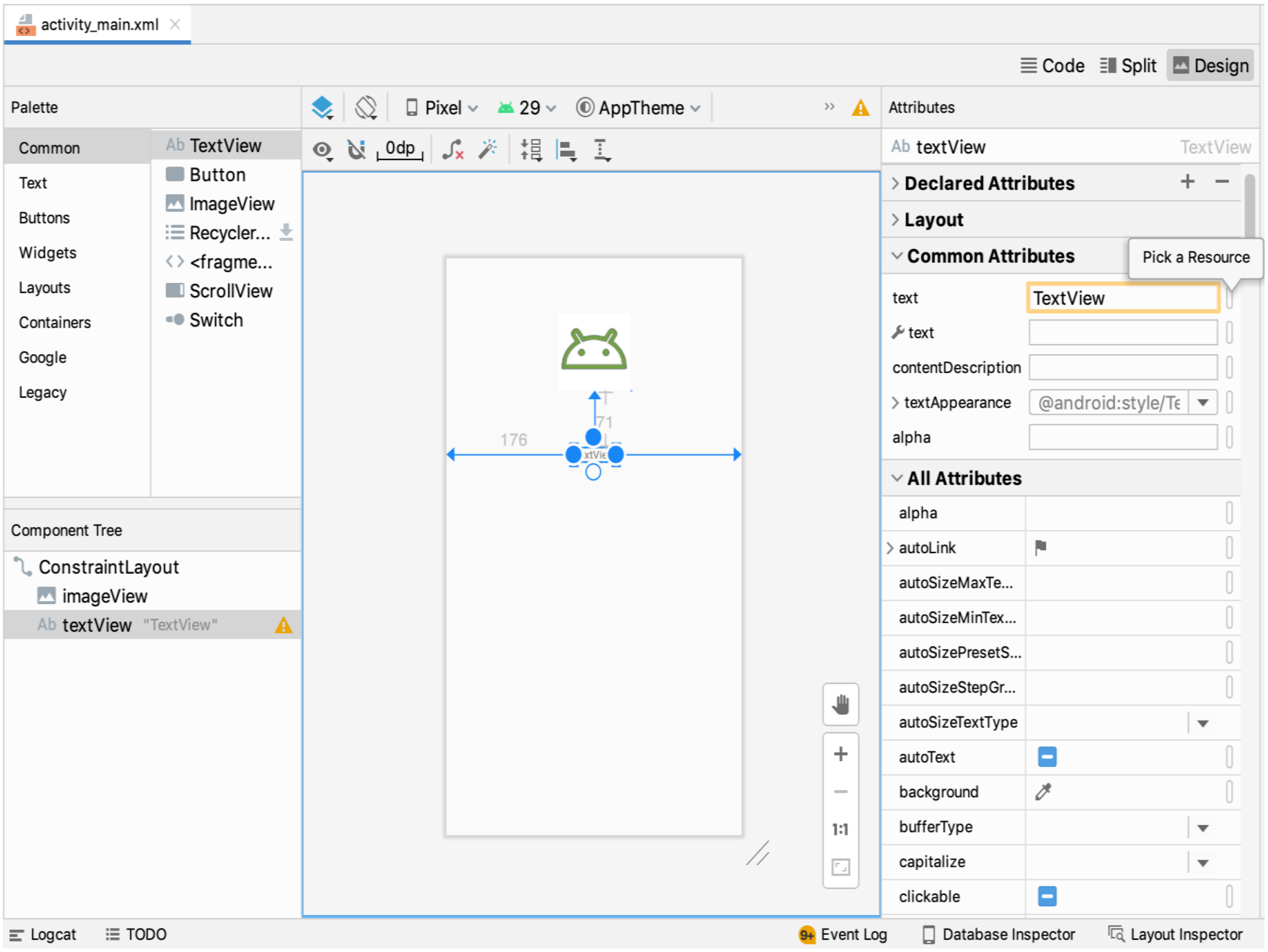
No (local only) | Preview | Cancel | Yes

Add text to the UI layout

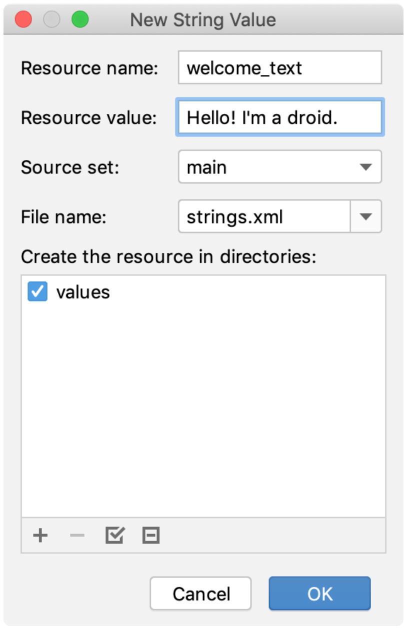
Now let's add some text to our layout.

1. In the Palette pane, pick the TextView element and drag it to the canvas below the image.
   The widget displays some default text: TextView. To change it and link it to a string, we need to create a new text resource.

2. Select the textView element in the Component Tree on the left. In the Attributes pane on the right, click the Pick a Resource icon next to the text attribute:



3. In the dialog that opens, click the Add resource to the module icon + in the top left corner and choose String Value.
4. In the New String Value dialog, enter the resource name (`welcome_text`) and the resource value (`Hello! I'm a droid.`):


New String Value

Resource name: welcome_text

Resource value: Hello! I'm a droid.

Source set: main

File name: strings.xml

Create the resource in directories:

☑ values

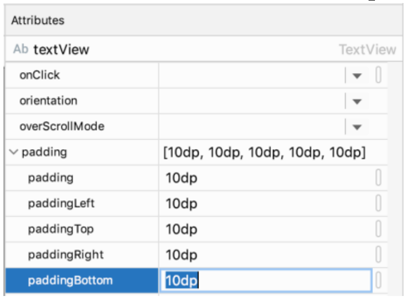
5. Click OK to save the value and then click OK in the Pick a Resource dialog.
6. Now let's modify the `textView` element id the same way we did with `imageView`.

Select `textView` in the Component Tree on the left, and in the Attributes pane set the id to a new value: `clickCounter`.

Add style to text

Now let's add some style to the text to make it look more appealing.

1. Pad the text a bit: locate the padding attribute, and set all values to `10dp`:



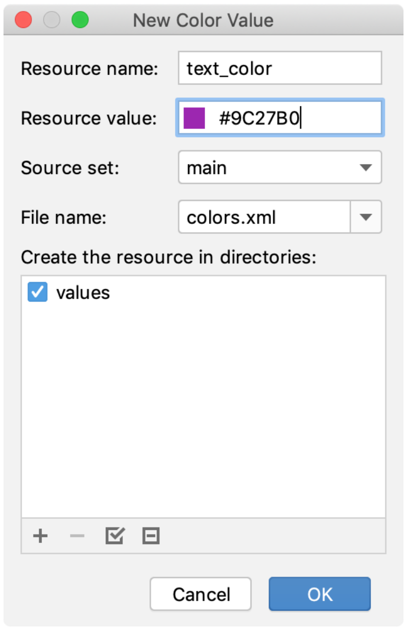
onClick	
orientation	
overScrollMode	
padding	[10dp, 10dp, 10dp, 10dp, 10dp]
padding	10dp
paddingLeft	10dp
paddingTop	10dp
paddingRight	10dp
paddingBottom	10dp

2. Change the font color: locate the textColor attribute, and click the Pick a

   Resource icon next to it.

   In the dialog that opens, click the Add resource to the module icon + in the top left corner and choose Color Value.

   Enter the resource name (`text_color`) and the value (`#9C27B0`):

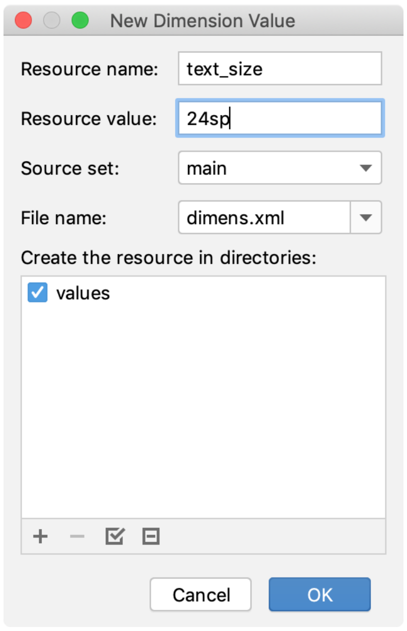


Resource name: text_color
Resource value: #9C27B0
Source set: main
File name: colors.xml
Create the resource in directories:
✓ values

3. Change the font size: locate the TextSize property and click the Pick a Resource icon next to it

   In the dialog that opens, click the Add resource to the module icon + in the top left corner and choose Dimension Value.

Enter the resource name (`text_size`) and the value (`24sp`):



New Dimension Value

Resource name: text_size

Resource value: 24sp

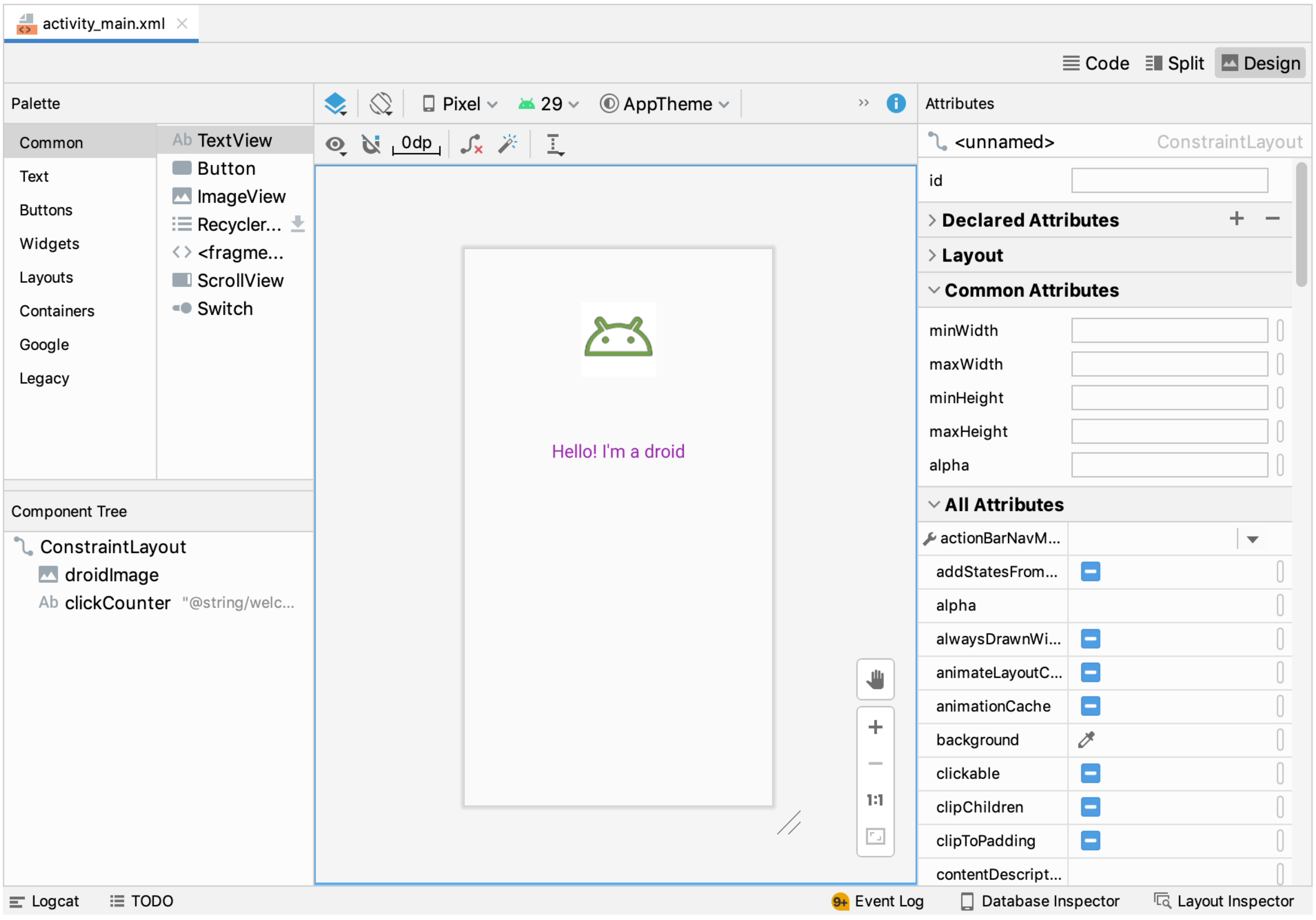
Source set: main

File name: dimens.xml

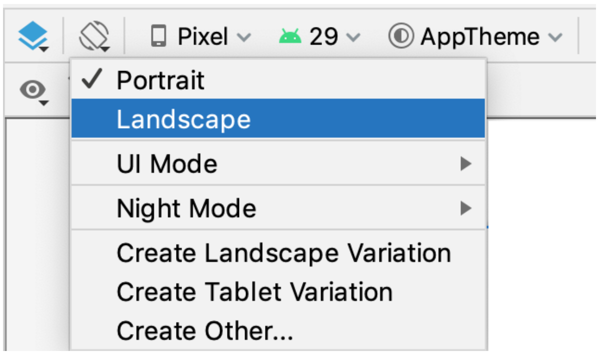
Create the resource in directories:

☑ values

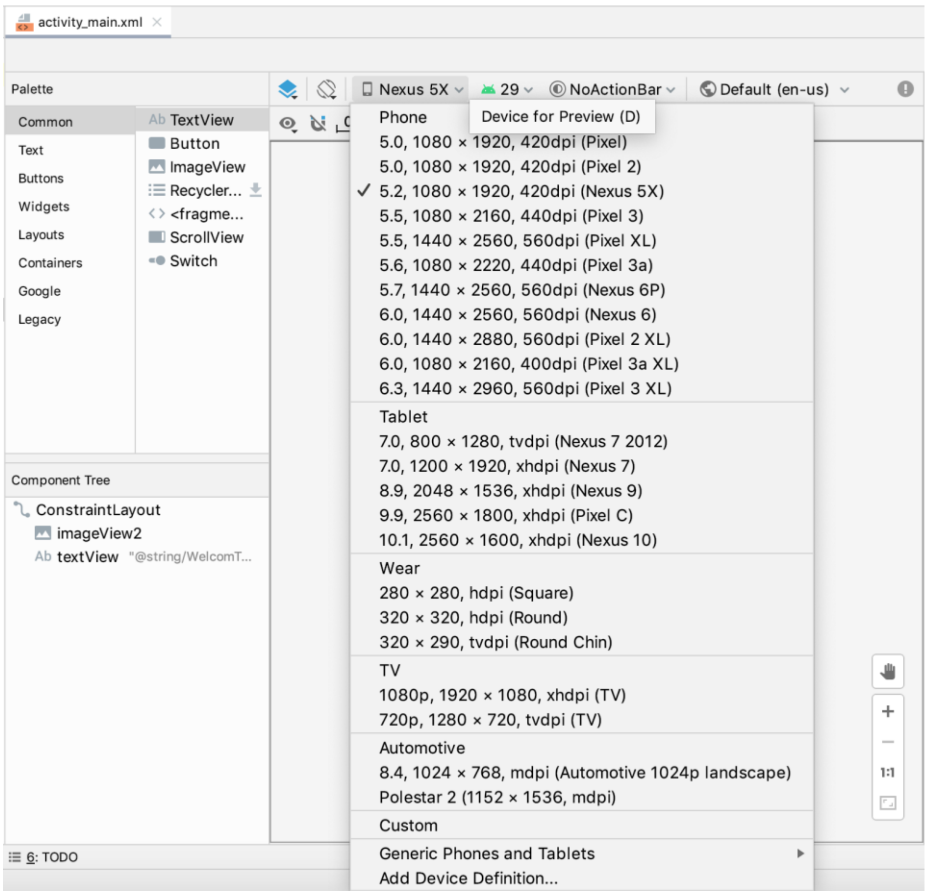
As a result, your user interface now looks like the following:



To check what your application UI looks like in landscape orientation, click the Orientation for Preview icon on the Designer toolbar and choose Landscape:



To preview what your layout looks like on different devices, select another device from the device list:

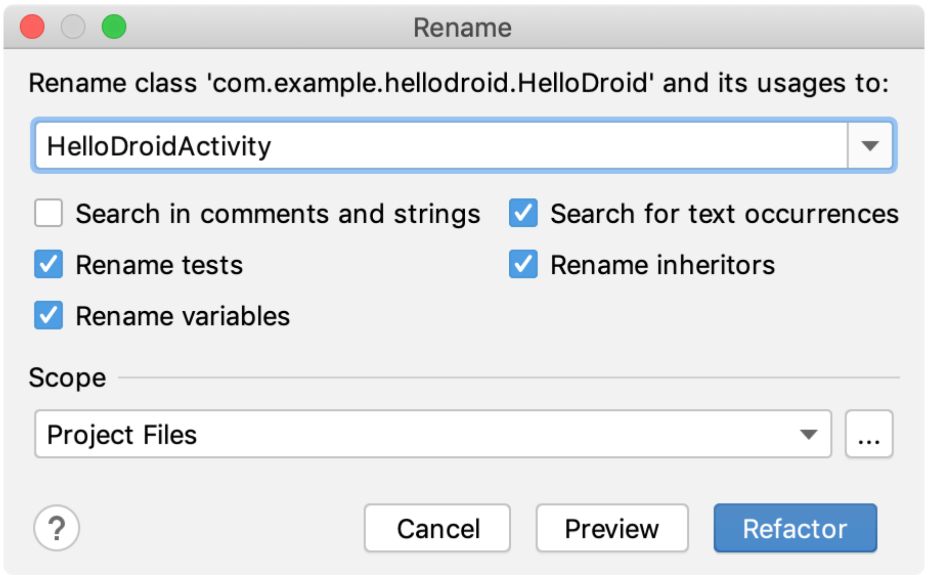

activity_main.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
Ab TextView
Button
ImageView
Recycler...
<fragme...
ScrollView
Switch
Component Tree
ConstraintLayout
imageView2
Ab textView "@string/WelcomT...
6: TODO
Nexus 5X
29
NoActionBar
Default (en-us)
Device for Preview (D)
Phone
5.0, 1080 × 1920, 420dpi (Pixel)
5.0, 1080 × 1920, 420dpi (Pixel 2)
✓ 5.2, 1080 × 1920, 420dpi (Nexus 5X)
5.5, 1080 × 2160, 440dpi (Pixel 3)
5.5, 1440 × 2560, 560dpi (Pixel XL)
5.6, 1080 × 2220, 440dpi (Pixel 3a)
5.7, 1440 × 2560, 560dpi (Nexus 6P)
6.0, 1440 × 2560, 560dpi (Nexus 6)
6.0, 1440 × 2880, 560dpi (Pixel 2 XL)
6.0, 1080 × 2160, 400dpi (Pixel 3a XL)
6.3, 1440 × 2960, 560dpi (Pixel 3 XL)
Tablet
7.0, 800 × 1280, tvdpi (Nexus 7 2012)
7.0, 1200 × 1920, xhdpi (Nexus 7)
8.9, 2048 × 1536, xhdpi (Nexus 9)
9.9, 2560 × 1800, xhdpi (Pixel C)
10.1, 2560 × 1600, xhdpi (Nexus 10)
Wear
280 × 280, hdpi (Square)
320 × 320, hdpi (Round)
320 × 290, tvdpi (Round Chin)
TV
1080p, 1920 × 1080, xhdpi (TV)
720p, 1280 × 720, tvdpi (TV)
Automotive
8.4, 1024 × 768, mdpi (Automotive 1024p landscape)
Polestar 2 (1152 × 1536, mdpi)
Custom
Generic Phones and Tablets
Add Device Definition...
1:1

Make the application interactive

Although our sample application is fully functional at this point, it does not support any form of interaction yet. Let's modify it to support tap events.

1. In the Android project view, locate the **MainActivity** file under **app\java\com.example.hellodroid** and double-click to open it.
2. **MainActivity** is not a very meaningful class name, so let's rename it.

   Right-click this file in the Android project view and choose Refactor | Rename from the context menu or press ShiftF6. In the dialog that opens, change the class name `HelloDroidActivity` and click Refactor:



Rename

Rename class 'com.example.hellodroid.HelloDroid' and its usages to:

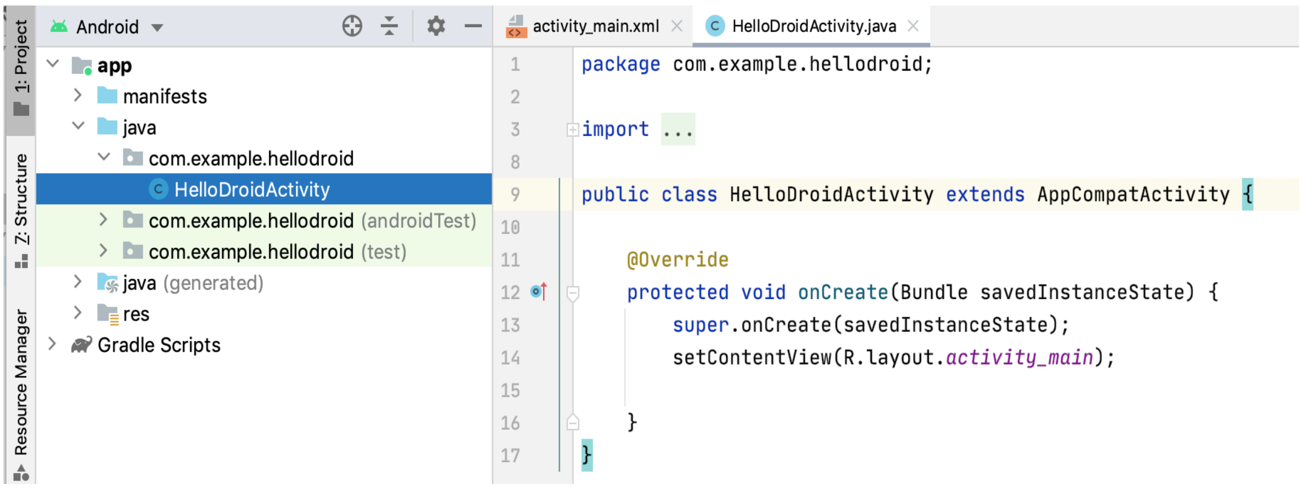
HelloDroidActivity

- [ ] Search in comments and strings
- [x] Search for text occurrences
- [x] Rename tests
- [x] Rename inheritors
- [x] Rename variables

Scope

Project Files

   All references to this class will be updated automatically, and your application's source code will look as follows:



```
package com.example.hellodroid;

import ...

public class HelloDroidActivity extends AppCompatActivity {

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

    }
}
```

3. Replace the code in **HelloDroid.java** with the following:

```
package com.example.hellodroid;


import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
```

```java
import androidx.appcompat.app.AppCompatActivity;

public class HelloDroidActivity extends AppCompatActivity {
private TextView message;
private int counter = 0;

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

        message = findViewById(R.id.clickCounter);
        ImageView droid = findViewById(R.id.droidImage);

        //Define and attach click listener
        droid.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                tapDroid();
            }
        });
    }

    private void tapDroid() {
        counter++;
        String countAsText;
        /*
         * In real applications you should not write switch like the one
below.
         * Use resource of type "Quantity strings (plurals)" instead.
         * See https://developer.android.com/guide/topics/resources/string-
resource#Plurals
         */
        switch (counter) {
            case 1:
                countAsText = "once";
                break;
            case 2:
                countAsText = "twice";
                break;
            default:
                countAsText = String.format("%d times", counter);
        }
        message.setText(String.format("You touched the droid %s",
countAsText));
    }
}
```

Note that the identifiers we've used in the source code correspond to those we've set in our layout definition file, otherwise our code would not work.

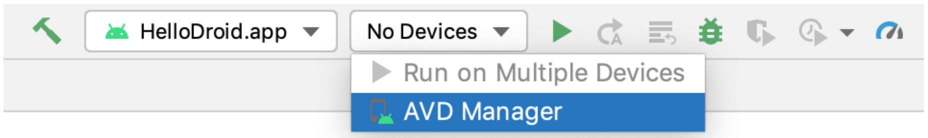
Build and run the application

Now let's build our application and run it on a virtual device.

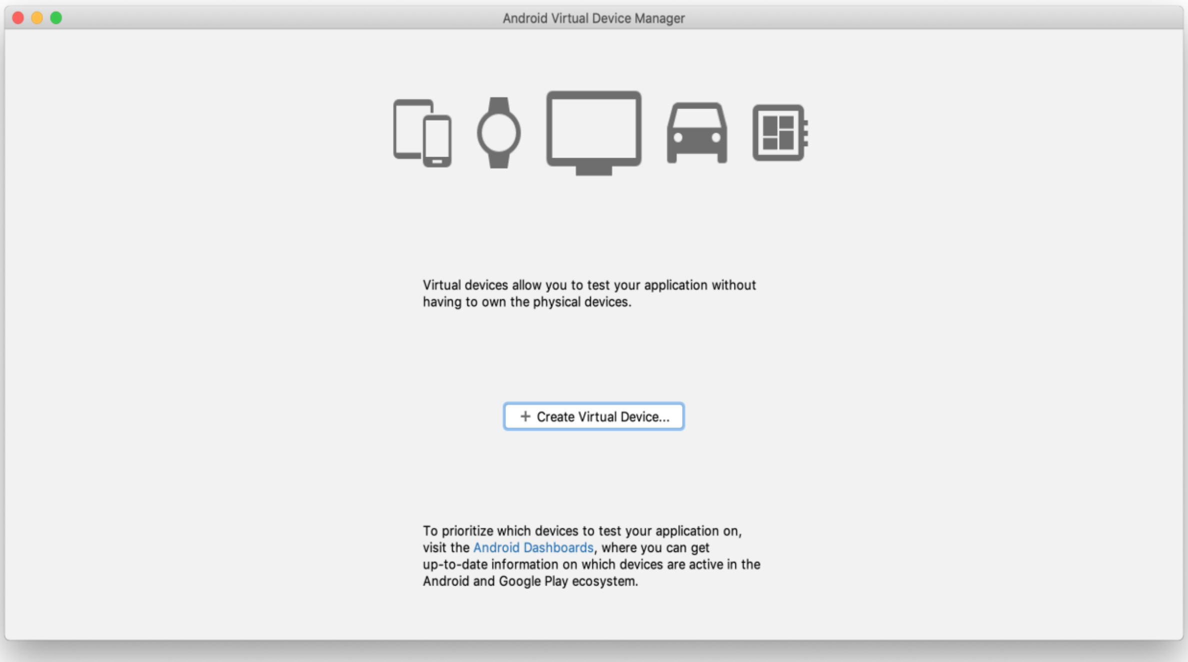
Configure Android virtual device

First of all, to be able to run our application, we need to configure a virtual device.

1. In the main IntelliJ IDEA toolbar, click the device list and choose AVD Manager:



2. On the first step of the wizard, click Create Virtual Device:

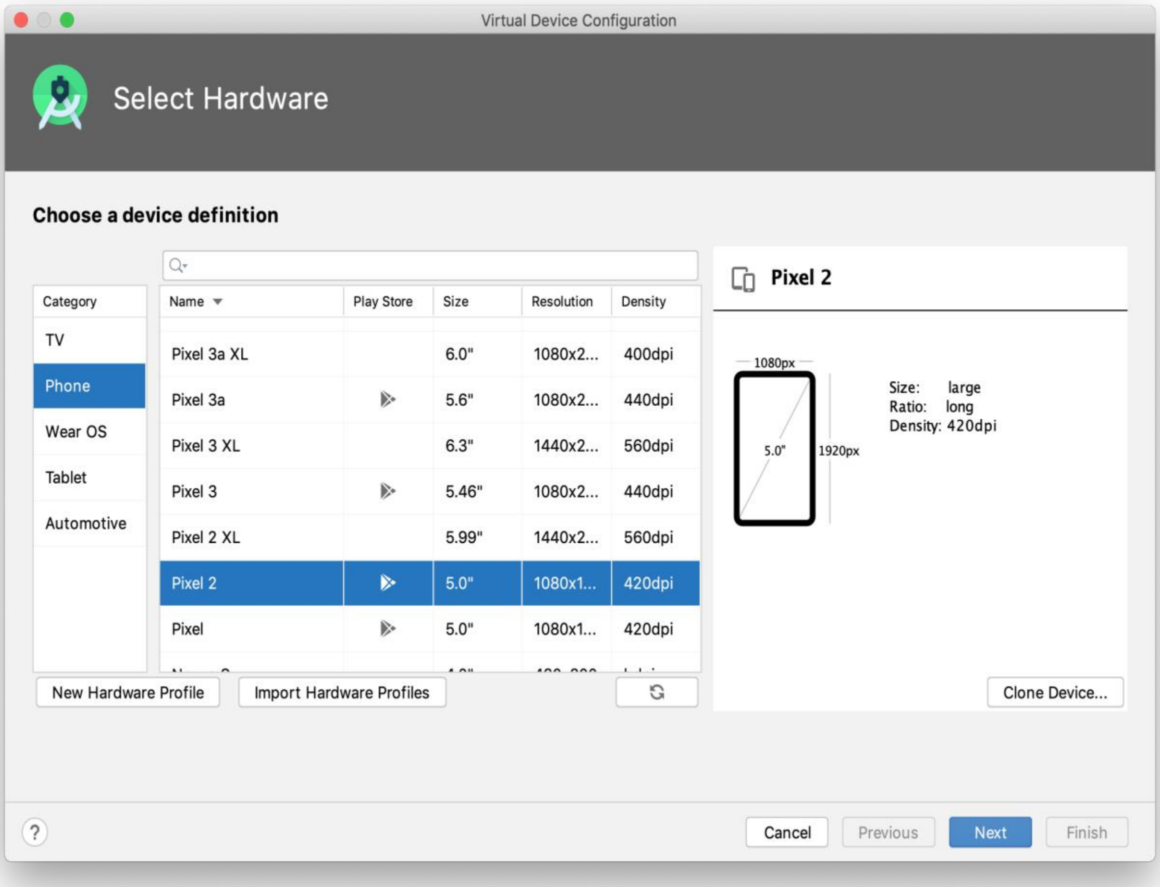


Virtual devices allow you to test your application without having to own the physical devices.

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.

3. On the next step, we need to select the hardware that our virtual device will emulate.

   Let's select Phone on the left, and choose Pixel 2 as the target device:


Virtual Device Configuration

Select Hardware

Choose a device definition

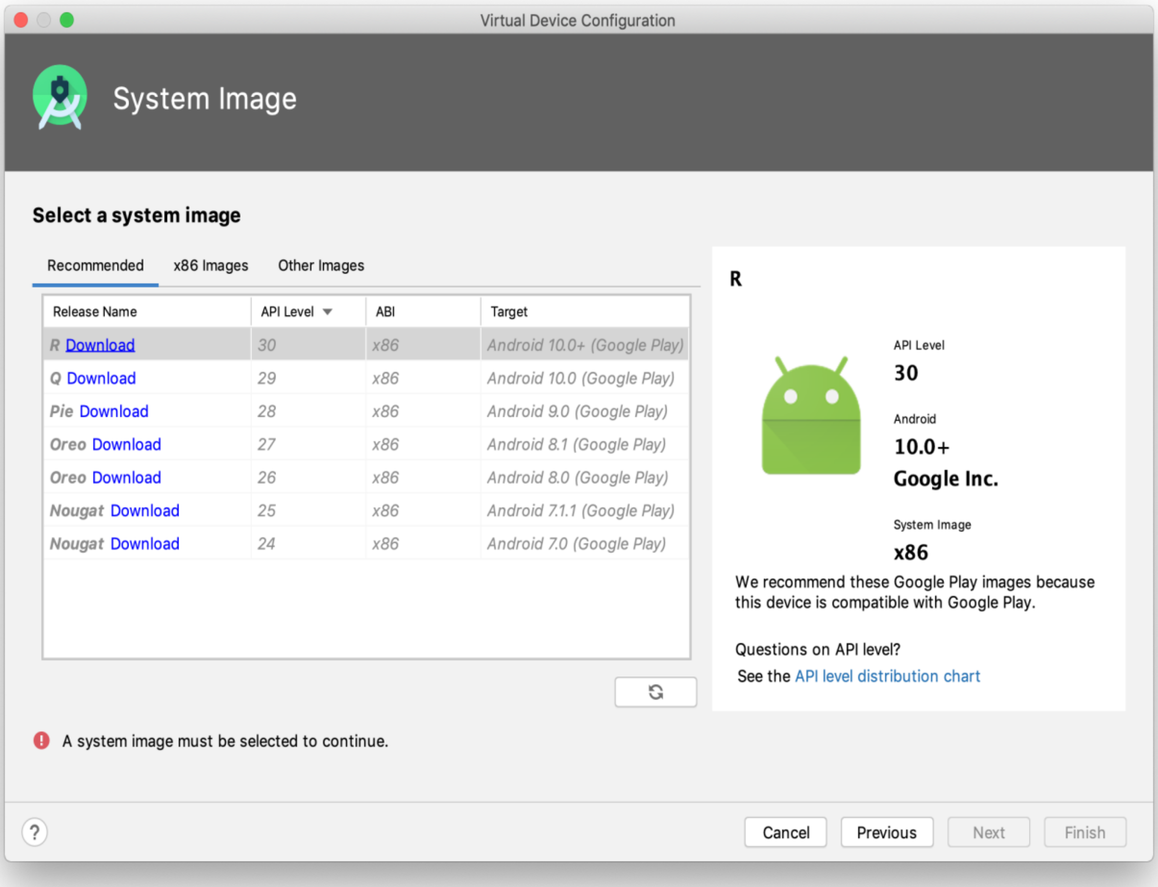
Category
TV
Phone
Wear OS
Tablet
Automotive

Name	Play Store	Size	Resolution	Density
Pixel 3a XL		6.0"	1080x2...	400dpi
Pixel 3a	▶	5.6"	1080x2...	440dpi
Pixel 3 XL		6.3"	1440x2...	560dpi
Pixel 3	▶	5.46"	1080x2...	440dpi
Pixel 2 XL		5.99"	1440x2...	560dpi
Pixel 2	▶	5.0"	1080x1...	420dpi
Pixel	▶	5.0"	1080x1...	420dpi

Pixel 2

Size: large
Ratio: long
Density: 420dpi

4. Choose the system image you want to mimic on the virtual device, that is the OS version, the Android API level, the application binary interface (ABI), and the target SDK version:


Virtual Device Configuration

System Image

Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
R Download	30	x86	Android 10.0+ (Google Play)
Q Download	29	x86	Android 10.0 (Google Play)
Pie Download	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

R

API Level
30

Android
10.0+
Google Inc.

System Image
x86

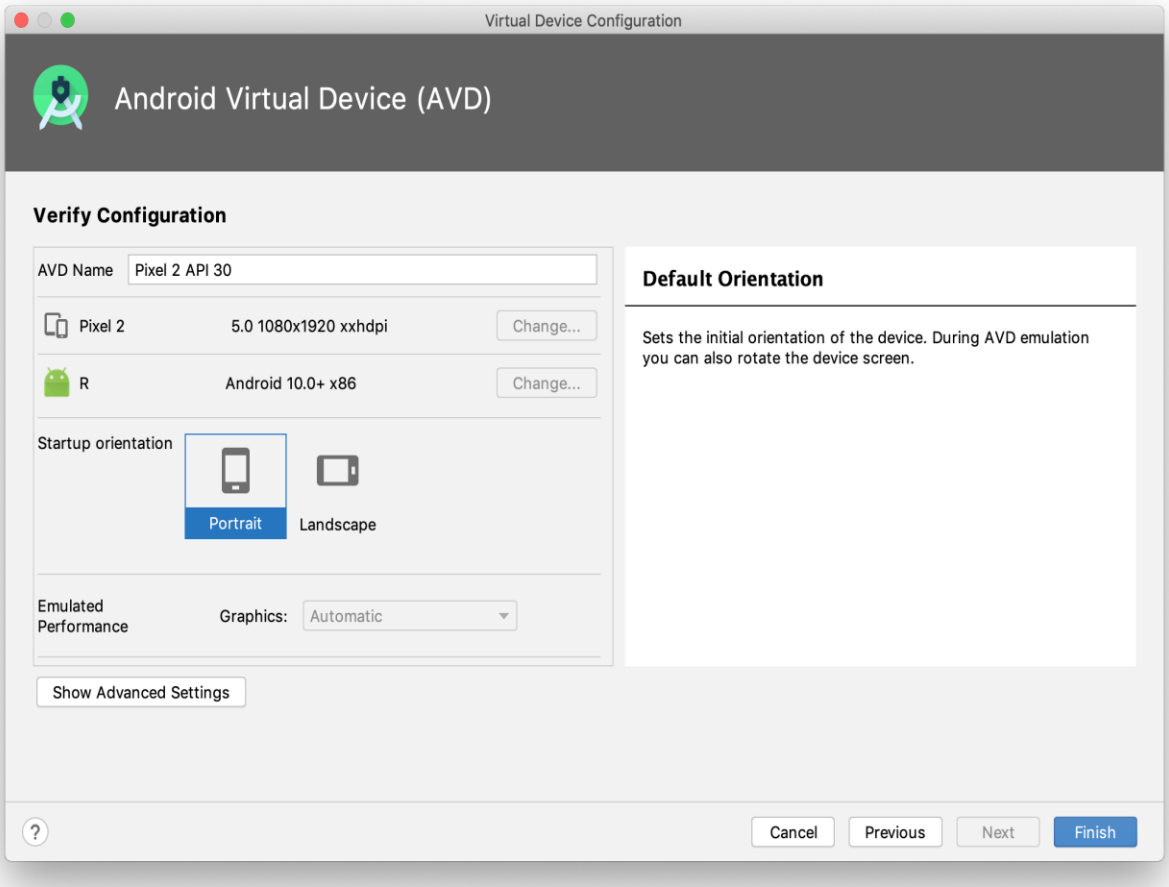
We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

A system image must be selected to continue.

5. Click the Download link next to the system image you want to mimic on the virtual device. For this tutorial, we've chosen to download the R system image.

   In the License Agreement dialog that opens, read the license agreement and accept it, then click Next and wait for the download to finish. When the system image has been downloaded, select it and click Next in the System Image step of the wizard.
6. On the last step, you can modify your virtual device name and select the startup size and orientation of the screen. Choose the portrait layout and click Finish:


Virtual Device Configuration

Android Virtual Device (AVD)

Verify Configuration

AVD Name: Pixel 2 API 30

Pixel 2 — 5.0 1080x1920 xxhdpi — Change...

R — Android 10.0+ x86 — Change...

Startup orientation: Portrait | Landscape

Emulated Performance — Graphics: Automatic

Show Advanced Settings

Default Orientation

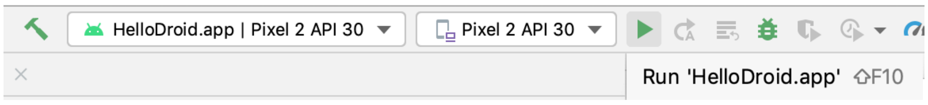
Sets the initial orientation of the device. During AVD emulation you can also rotate the device screen.

Cancel | Previous | Next | Finish

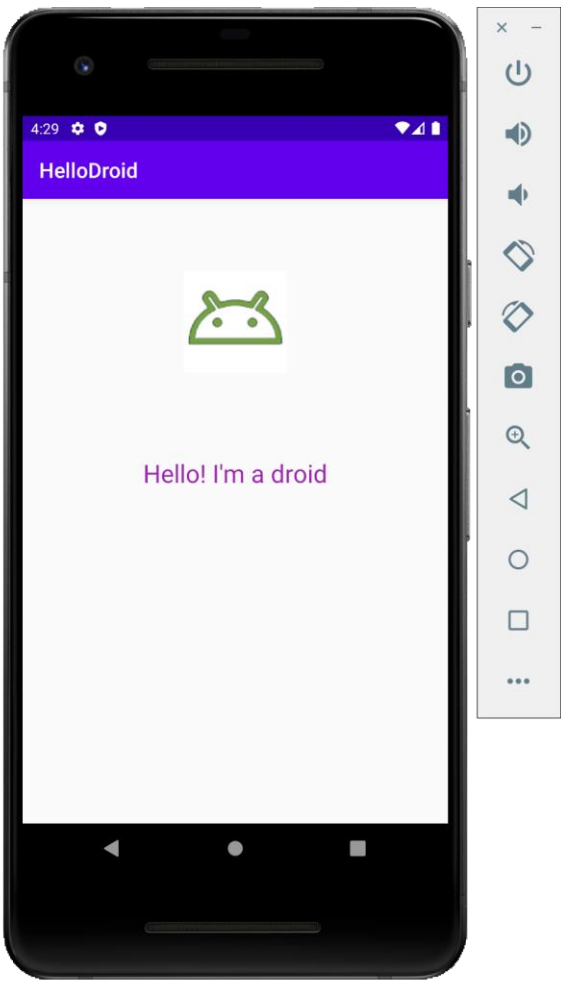
7. The newly configured device appears in the Android Virtual Device Manager.

Run the application

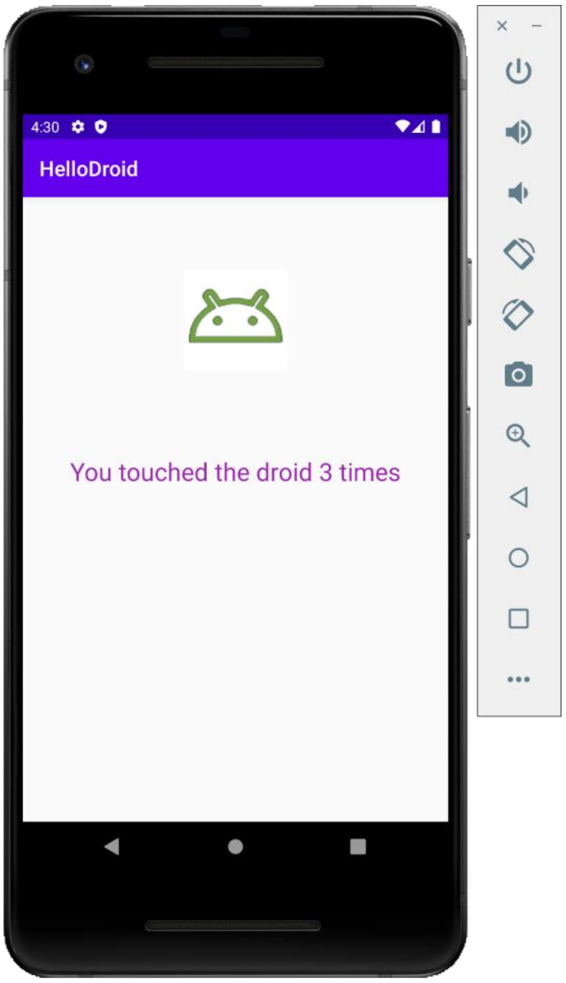
1. On the main IntelliJ IDEA toolbar, make sure the automatically created Run configuration and the virtual device we've just configured are selected and click :



The Android emulator will launch after the build has successfully finished, with our application started:



2. Click the droid image and see how the application processes the tap events, counts them and returns the corresponding message:



You touched the droid 3 times