ANDROID PROGRAMMING COOKBOOK

Kick-start your Android projects

CHRYSSA ALIFERI

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Android Programming Cookbook

Contents

Preface

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input.

In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Android has the largest installed base of all operating systems of any kind. Android has been the best selling OS on tablets since 2013, and on smartphones it is dominant by any metric. (Source: https://en.wikipedia.org/wiki/Android_(operating_system))

In this ebook, we provide a compilation of Android programming examples that will help you kick-start your own web projects. We cover a wide range of topics, from Services and Views, to Google Maps and Bluetooth functionality. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

Chryssa is a Computer Science graduate from Athens University of Economic and Business. During her studies, Chryssa carried out a great variety of projects ranging from networking to software engineering.

She is very keen on front end development especially on mobile technologies and web applications. She has worked as a junior Software Engineer in the telecommunications area and currently works as an Android Developer.

Chapter 1

Android Tutorial For Beginners

There are lots of reasons why more and more people are interested in learing how to be able to develop Android applications. Unarguably, Android is the **most popular mobile operating system**, with almost 2 billion devices activated and it offers a unified approach to application development for mobile devices.

That means, that developers need only develop for Android, and their applications will be able to run on different devices powered by Android. This particular asset gives Android endless possibilites! This means that and application that is designed to work on mobile phone devices can be also transfered to Android powered TV sets or Android Car systems.

This is why, Android is an exciting space to make apps that can help you in every aspect of your life, can help you communicate, organize, educate, entertain or just to make your life easier in every device that they might run on!

In this special example, we are going to set our **Android Development Studio IDE**, make our very **first Android application** and discover the **Android Development world** in the easiest possible way.

The mobile development world can be very fun, because the direct results we see when creating our own application, can be highly motivating and rewarding.

1.1 What is Android?

Android is a mobile operating system currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. And as we said before, Android offers a unified approach to application development for mobile devices.

Android is an open-source operating system named Android. Google has made the code for all the low-level "stuff" as well as the needed middleware to power and use an electronic device, and gave Android freely to anyone who wants to write code and build the operating system from it. There is even a full application framework included, so third-party apps can be built and installed, then made available for the user to run as they like.

The "proper" name for this is the Android Open Source Project, and this is what people mean when they say things like Android is open and free. Android, in this iteration, is free for anyone to use as they like.

1.2 Installing Android Studio

In order to write an Android application, we are going to need a development environment. Google has made a very useful tool for all Android Developers, the **Android Studio**. Android Studio is the official IDE for Android development, and with a single download includes everything you need to begin developing Android apps. Included in the download kit, are the **Software Development Kit (SDK)**, with all the **Android libraries** we may need, and the infrastructure to download the many **Android emulator** instances, so that we can initially run our application, without needing a real device.

So, we are going to download and install Android Studio.

First we have to have installed the Java Development Kit (JDK) from Oracle. If you do not, please you should download the latest JDK from the **Oracle's special section** here.

Figure 1.1: Android Studio Installation - step 1

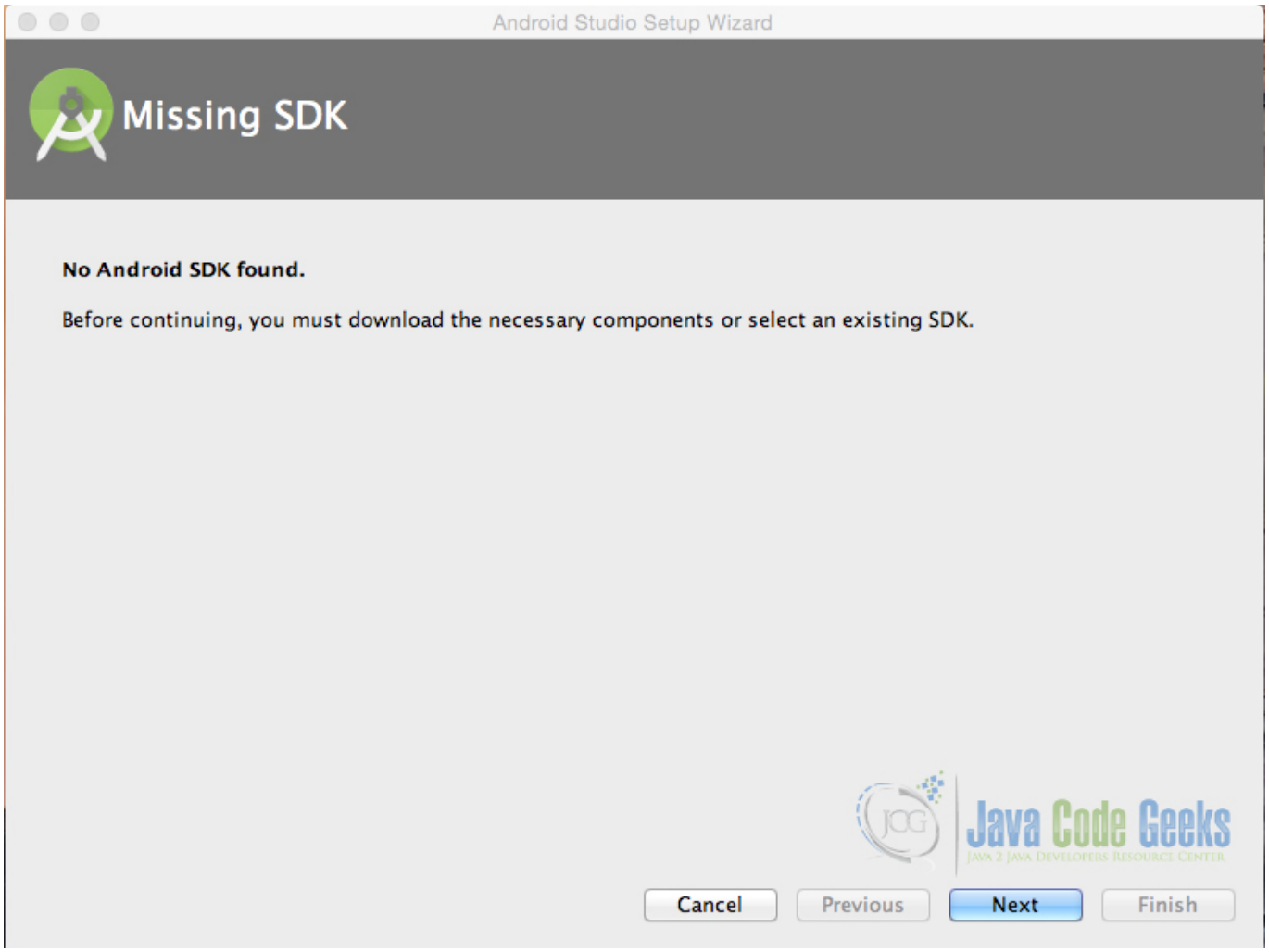

Figure 1.2: Android Studio Installation - step 2

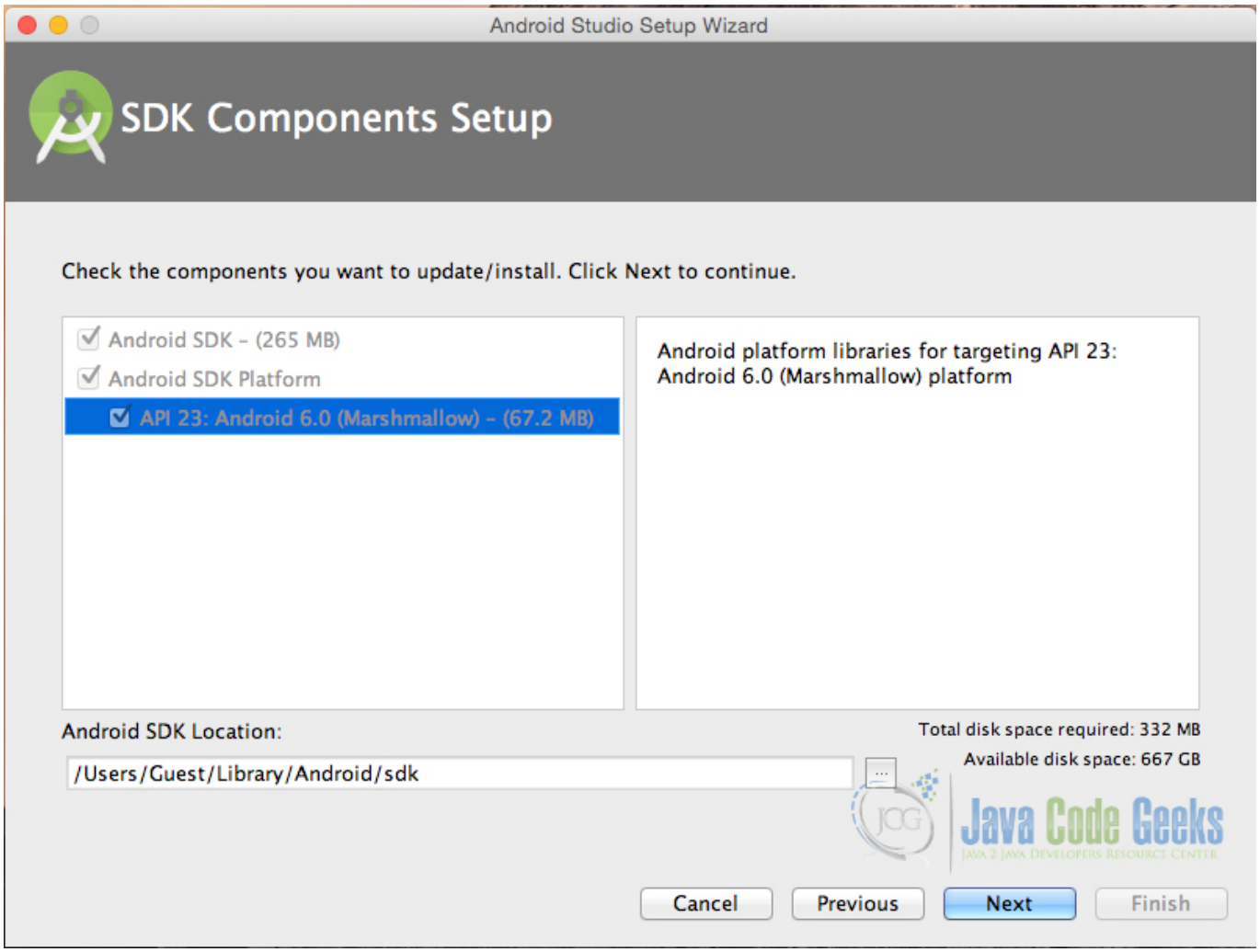

Figure 1.3: Android Studio Installation - step 3

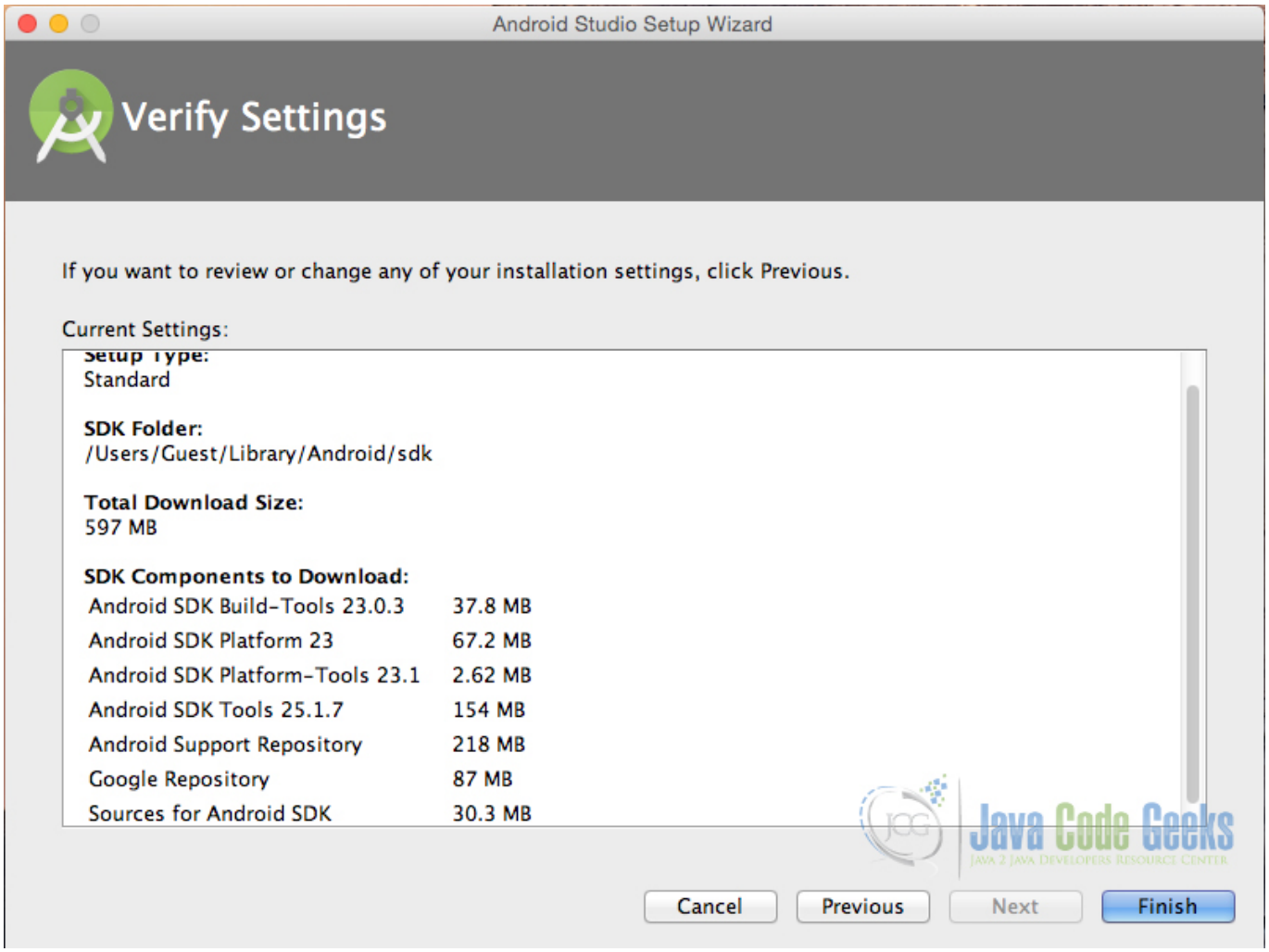

Figure 1.4: Android Studio Installation - step 4

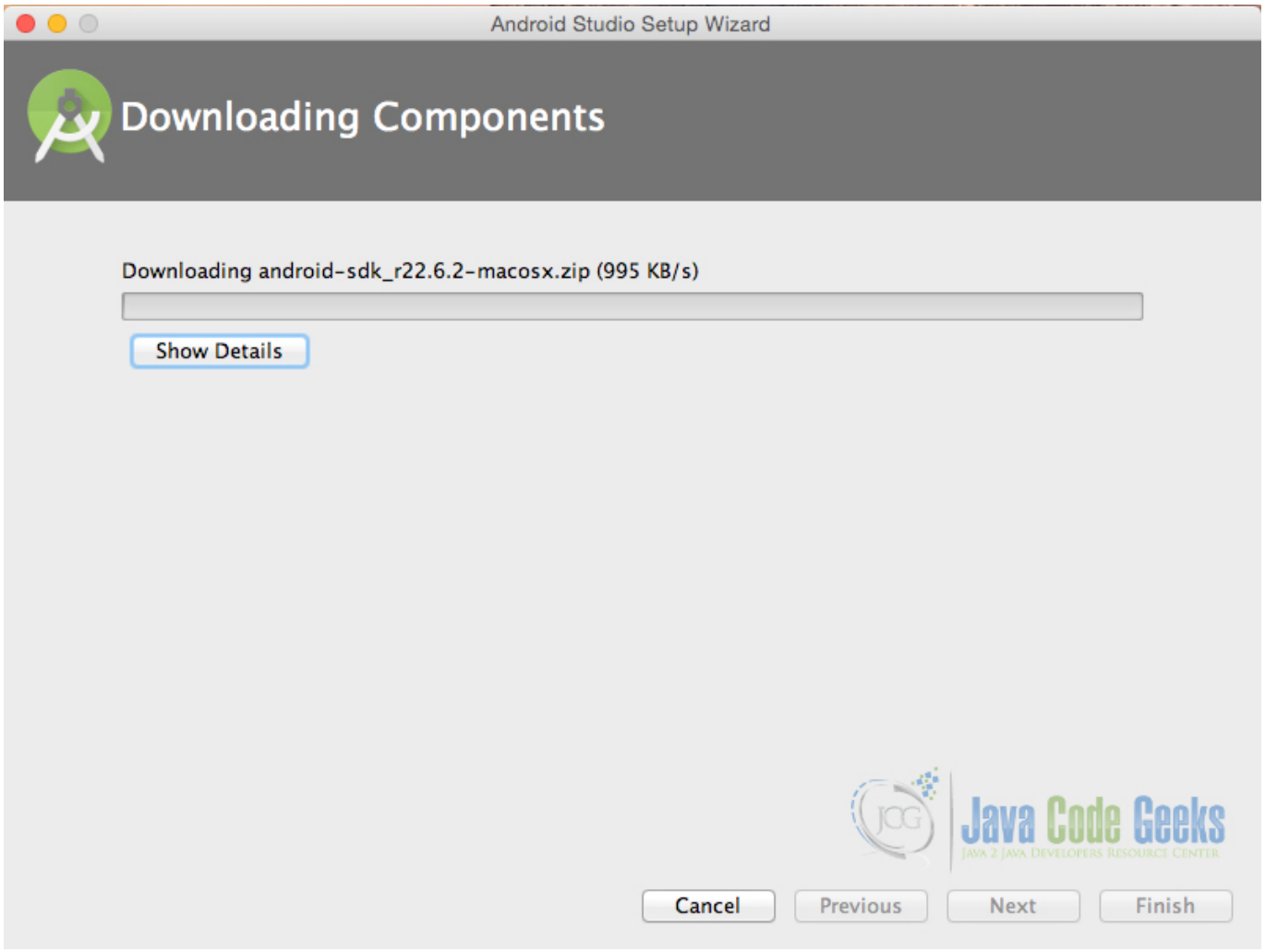

Figure 1.5: Android Studio Installation - step 5

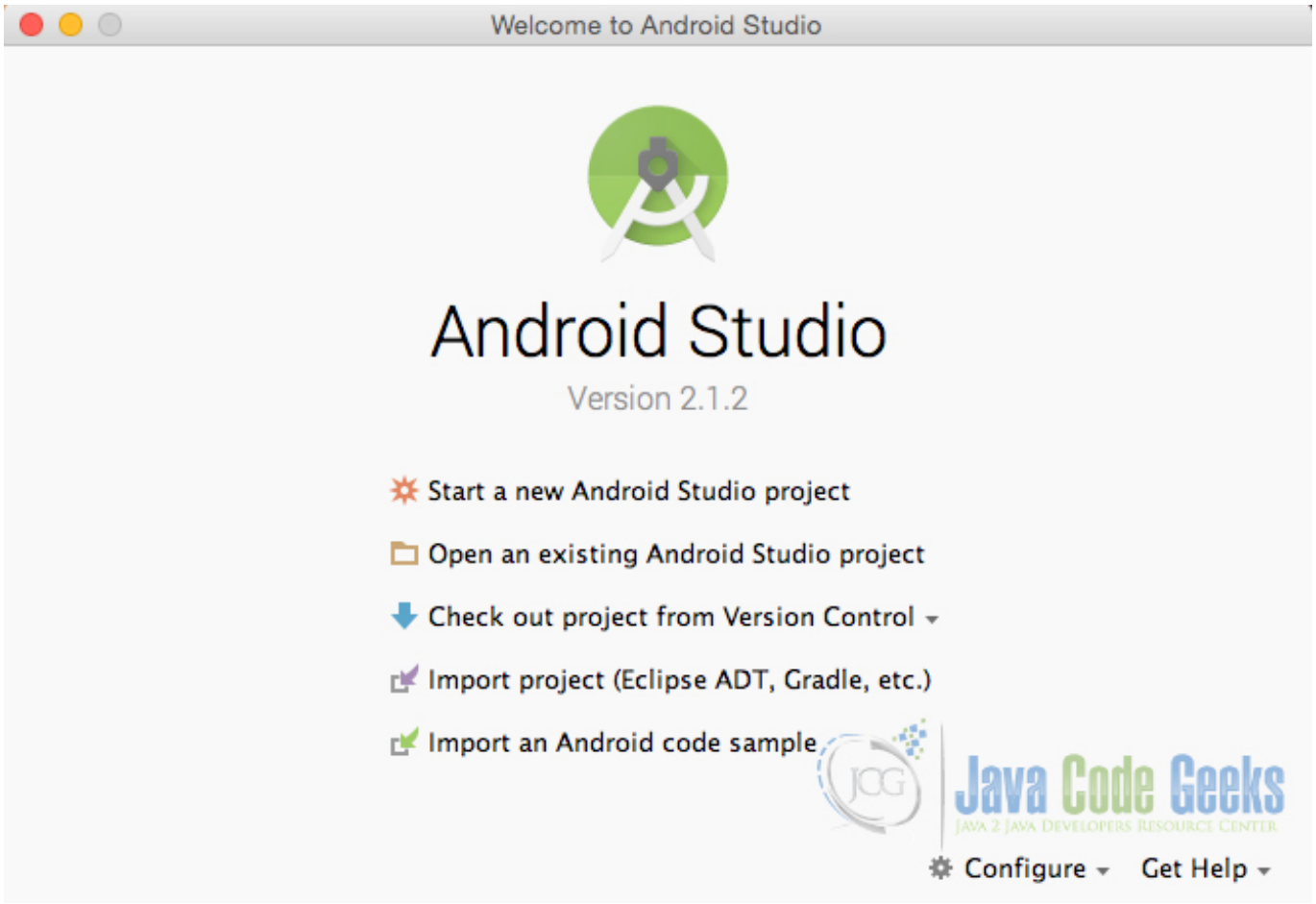

Figure 1.6: Android Studio Installation - Ready

1.3 Android versions and Android SDK Manager

Google, releases almost every year (or even sooner than a year), a **new Android version** in order to update the mobile operating system, so that it contains **new features and possibilities** and of course to fix things that might not work in the right way.

So, each version of Android has it's own **SDK** (software development kit), so that we can use to build apps that can run on and include all the latest features Android has added in its latest versions.

This means that it is essential that we keep up updating our applications with the latest features all the time, and if possible, without losing the consistency of the previous Android versions.

As part of the Setup Wizard you will already have the latest SDK available to you, however it's useful to know how to install additional SDK's if you need to work with older devices that do not run the latest version of Android.

SDK's allow you to create AVD's (Android Virtual Devices) to test your Apps on, customized to your personal configuration. Want to see how your Android App looks on a TV sized screen? If you have a screen big enough you can find out.

1.4 Supporting different screen sizes

Android runs on a variety of devices that offer different **screen sizes and densities**. This means that Android can handle applications that run on small mobile phone devices, as well as applications that run on large tablet densities.

This feature gives Android a great advantage, but also, although the system performs scaling and resizing on different screens, developers should make the effort to optimize their application for different screen sizes and densities. Android system provides a consistent development environment across devices and handles most of the work to adjust each application's user interface to the screen on which it is displayed. At the same time, the system provides APIs that allow you to control your application's UI for specific screen sizes and densities, in order to optimize your UI design for different screen configurations. For example, you might want a UI for tablets that's different from the UI for handsets.

Below is an introduction to the terms and concepts used, a summary of the screen configurations that the system supports, and an overview of the API and underlying screen-compatibility features:

Screen size Actual physical size, measured as the screen's diagonal. For simplicity, Android groups all actual screen sizes into four generalized sizes: small, normal, large, and extra-large.

Screen density The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen. For simplicity, Android groups all actual screen densities into six generalized densities: low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high.

Orientation The orientation of the screen from the user's point of view. This is either landscape or portrait, meaning that the screen's aspect ratio is either wide or tall, respectively. Be aware that not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.

Resolution The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

Density-independent pixel (dp) A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way. The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen. At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple: px = dp * (dpi / 160). For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

Android provides support for multiple screen sizes and densities, reflecting the many different screen configurations that a device may have. You can use features of the Android system to optimize your application's user interface for each screen configuration and ensure that your application not only renders properly, but provides the best user experience possible on each screen.

To simplify the way that you design your user interfaces for multiple screens, Android divides the range of actual screen sizes and densities into sizes: small, normal, large, and xlarge

A set of six generalized densities:

- **ldpi** (low) ~120dpi
- **mdpi** (medium) ~160dpi
- **hdpi** (high) ~240dpi
- **xhdpi** (extra-high) ~320dpi
- **xxhdpi** (extra-extra-high) ~480dpi
- **xxxhdpi** (extra-extra-extra-high) ~640dpi

1.5 Android Project Structure

Before we try to make our first Android application, we should first see the basic parts of an Android application project, in order to recognize them and be able to understand them better.

- **Activities** The Activities are the main Java classes, that contain the Android code with which we are going to develop, what do we want the application to do.

- **Layouts** The Layouts are the main xml files, that contain the Android xml code with which we are going to develop, how will our application views look like.
- **Values** The Layouts are the main xml files, that contain the Android xml code with which we are going to develop, how will our application views look like.
  - Animation Resources
  - Color State List Resource
  - Drawable Resources
  - Layout Resource
  - Menu Resource
  - String Resources
  - Style Resource
- **Drawables** A drawable resource is a general concept for a graphic that can be drawn to the screen. There are several different types of drawables:
  - Bitmap File A bitmap graphic file (.png, .jpg, or .gif). Creates a BitmapDrawable.
  - Nine-Patch File A PNG file with stretchable regions to allow image resizing based on content (.9.png). Creates a NinePatchDrawable.
  - Layer List A Drawable that manages an array of other Drawables. These are drawn in array order, so the element with the largest index is be drawn on top. Creates a LayerDrawable.
  - State List An XML file that references different bitmap graphics for different states (for example, to use a different image when a button is pressed). Creates a StateListDrawable.
  - Level List An XML file that defines a drawable that manages a number of alternate Drawables, each assigned a maximum numerical value. Creates a LevelListDrawable.
  - Transition Drawable An XML file that defines a drawable that can cross-fade between two drawable resources. Creates a TransitionDrawable.
  - Inset Drawable An XML file that defines a drawable that insets another drawable by a specified distance. This is useful when a View needs a background drawble that is smaller than the View's actual bounds.
  - Clip Drawable An XML file that defines a drawable that clips another Drawable based on this Drawable's current level value. Creates a ClipDrawable.
  - Scale Drawable An XML file that defines a drawable that changes the size of another Drawable based on its current level value. Creates a ScaleDrawable
  - Shape Drawable An XML file that defines a geometric shape, including colors and gradients. Creates a ShapeDrawable.

Once our app is ready, we will use a build tool to compile all the project files and package them together into an .apk file that you can run on Android devices and/or submit to Google Play.

1.6 Create "Hello Android World" application

1.6.1 Create a New Android Studio Project

Open Android Studio and choose Start a new Android Studio Project in the welcome screen.

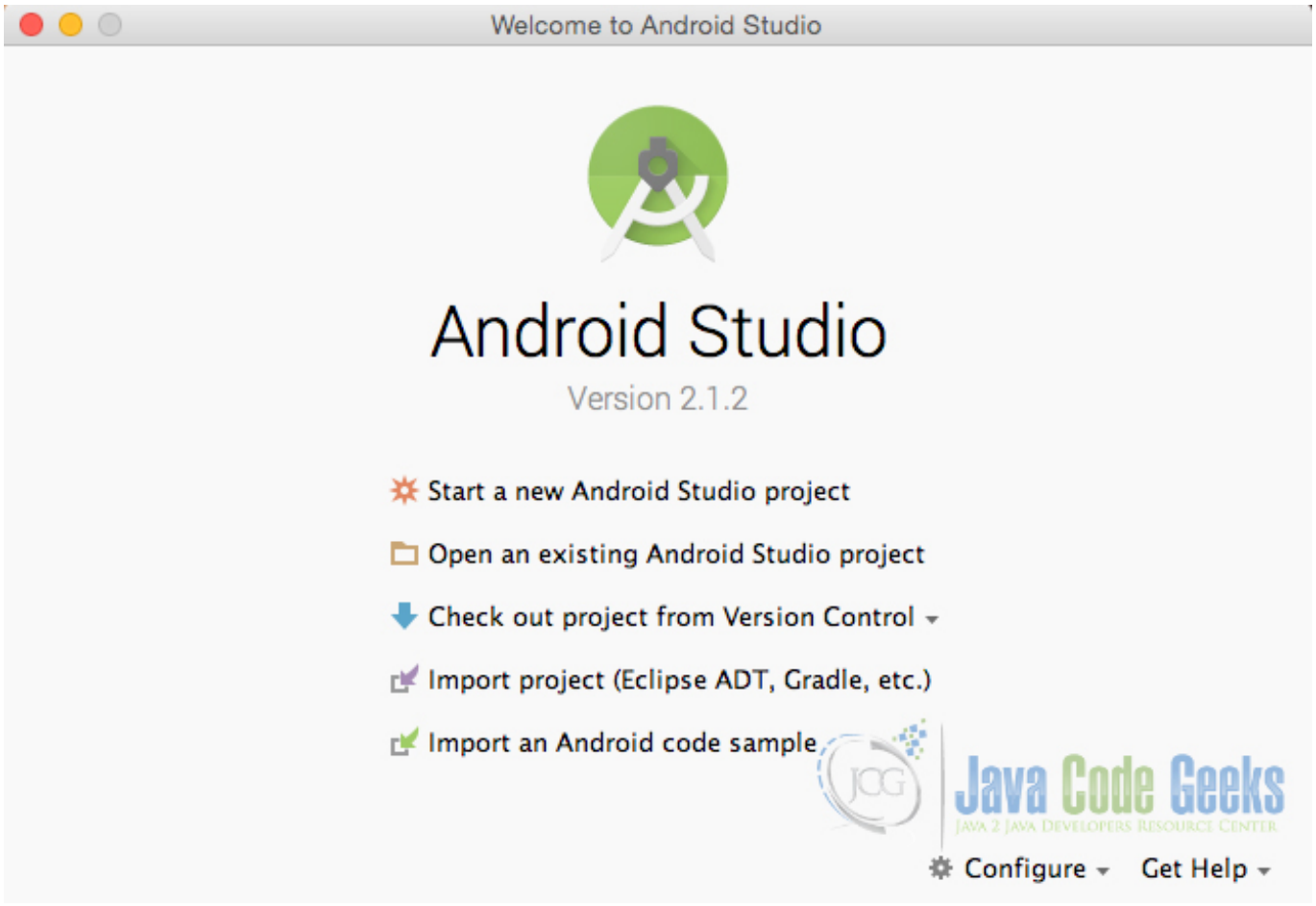

Figure 1.7: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

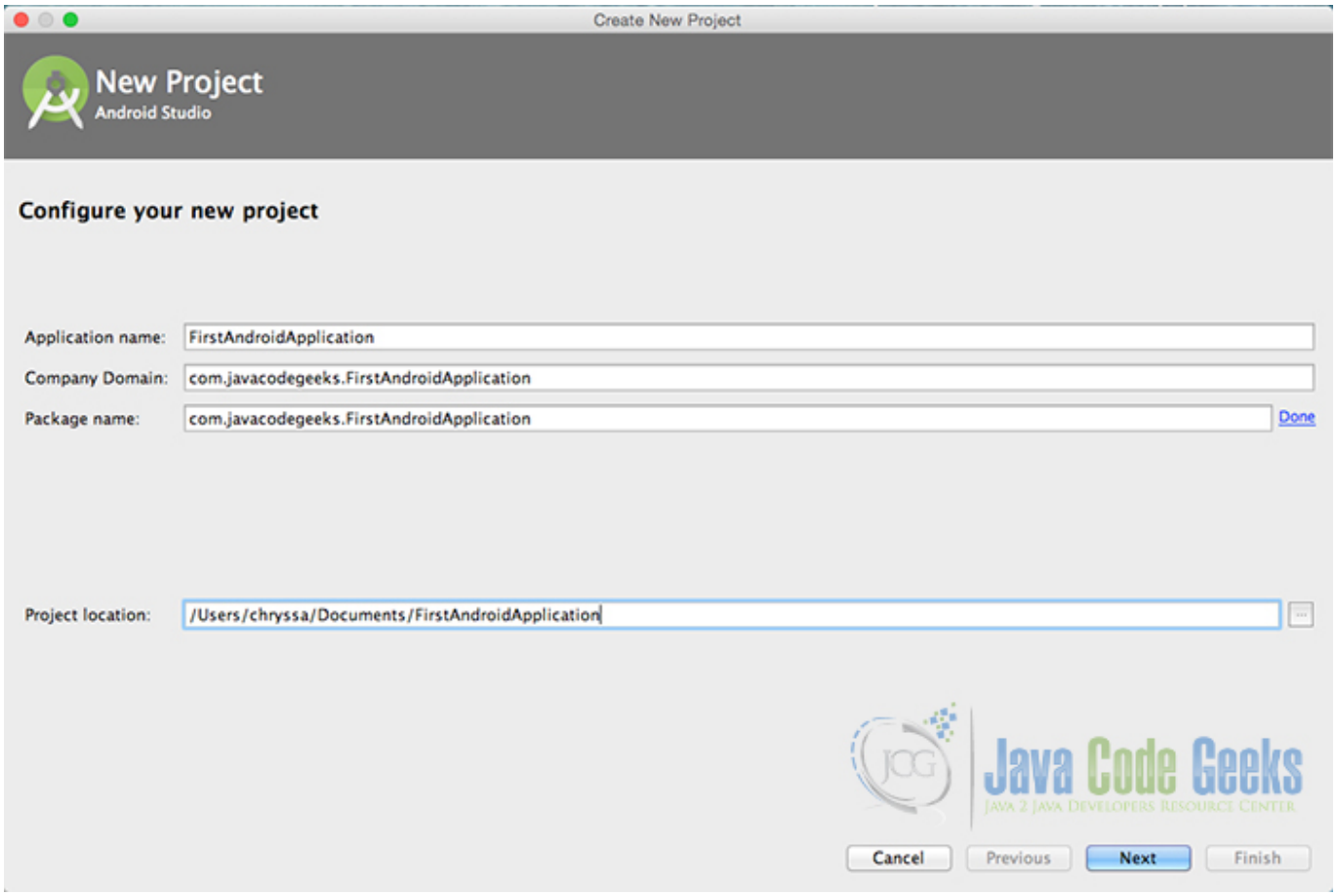

Figure 1.8: Configure your new project screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

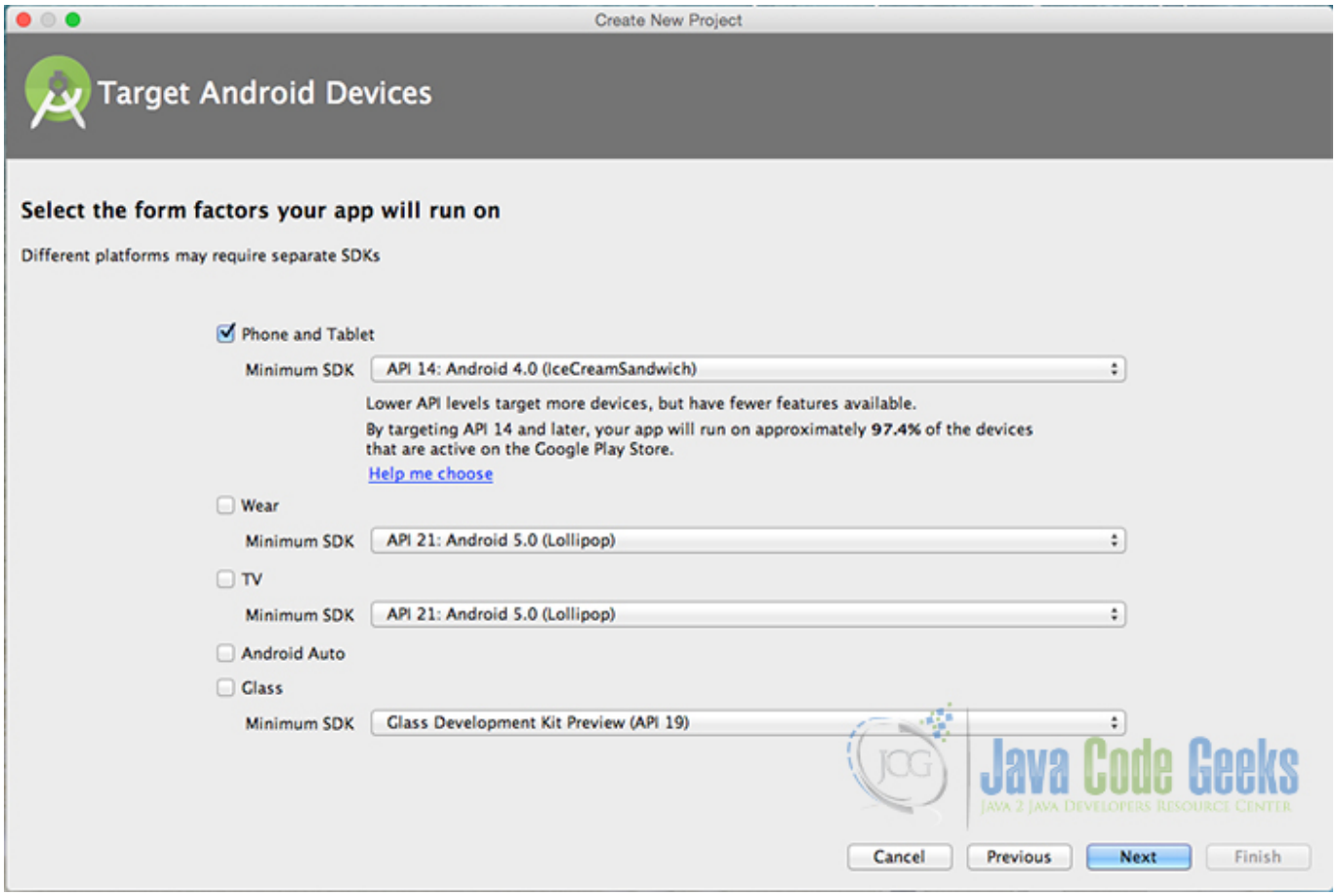

Figure 1.9: Target Android Devices screen.

In the next window you should choose Add no activity. In this example, we are going to create our Activity.

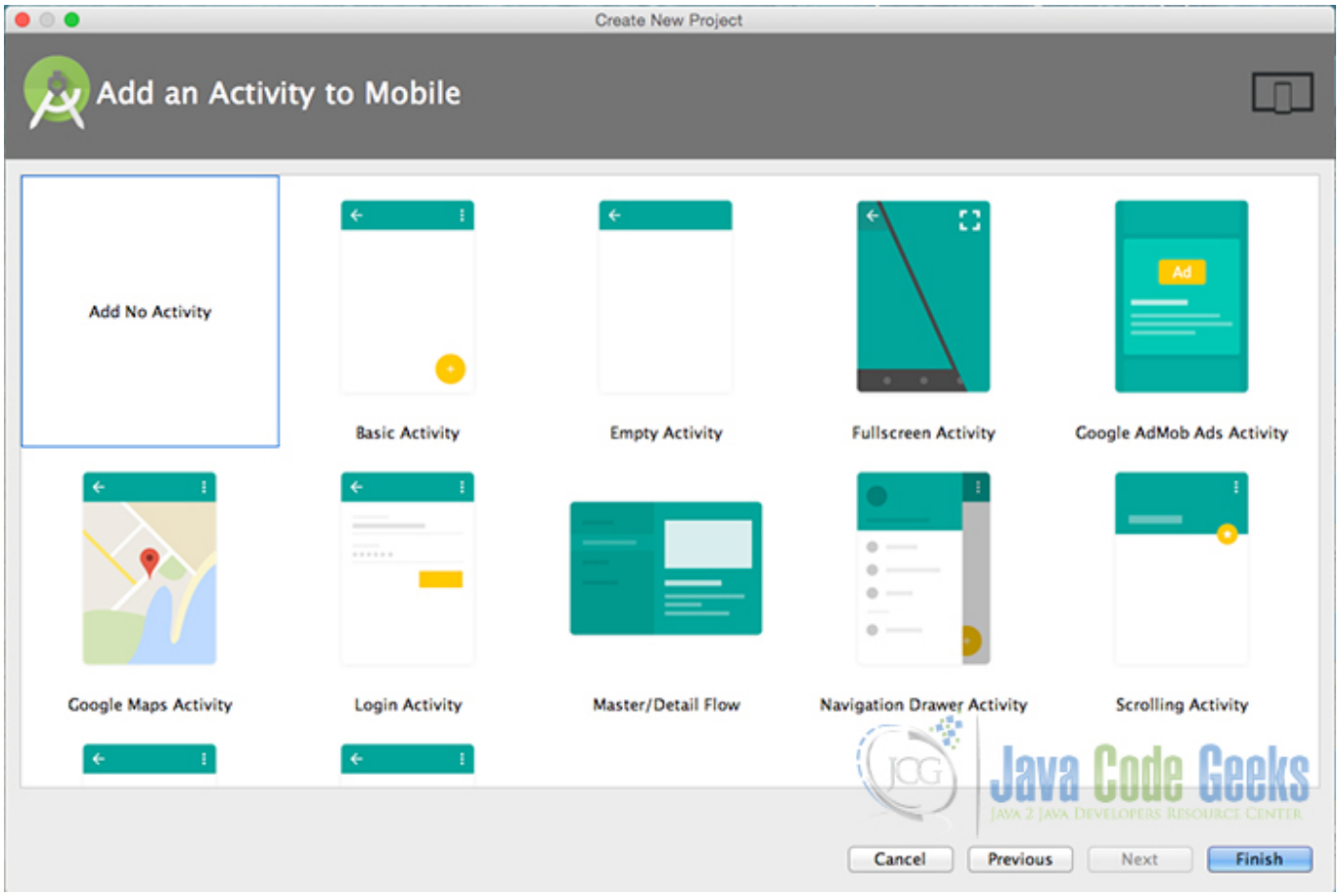

Figure 1.10: Add an activity to Mobile. Choose: Add no activity.

Now, our project has just been created!

1.6.2 Create the source code of a simple FirstAndroidApplication Activity

Add a new Java class Activity inside `src/com.javacodegeeks.FirstAndroidApplication/` so that we are going to have the `src/com.javacodegeeks.FirstAndroidApplication/FirstActivity.java` file and paste the code below.

FirstActivity.java

```
package com.javacodegeeks.FirstAndroidApplication;

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

public class FirstActivity extends Activity {

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

}
```

1.6.3 Create the layout of the project

Add a new xml file inside `/res/layout` folder, with name `main_layout.xml`. We should have the `/res/layout/main_layout.xml` file and paste the code below.

main_layout.xml

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="@dimen/textmargin"
        android:gravity="center"
        android:textSize="25dp"
        android:text="@string/helloAndroid" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/logomargin"
        android:background="@drawable/ic_social_mood" />

</LinearLayout>
```

1.6.4 Android Manifest

Edit the `AndroidManifest.xml` file inside `/app/manifests` folder. The AndroidManifest.xml of our project is simple and should be like this:

AndroidManifest.xml

```
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.FirstAndroidApplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".FirstActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

1.6.5 Edit the FirstAndroidApplication dimensions

Add a new xml file inside `/res/values` folder, with name `dimens.xml`. We should have the `/res/values/dimens.xml` file and paste the code below.

dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="logomargin">20dp</dimen>
    <dimen name="textmargin">10dp</dimen>
</resources>
```

1.6.6 Edit the FirstAndroidApplication strings

Add a new xml file inside `/res/values` folder, with name `strings.xml`. We should have the `/res/values/strings.xml` file and paste the code below.

strings.xml

```
<resources>
    <string name="app_name">AndroidFirstApplication</string>
    <string name="helloAndroid">Hello Android!</string>
</resources>
```

1.6.7 Add the drawable for every screen density

Inside `/res/values` folder, we should add the folders for each screen dimension we have, and add the specific drawable for each one.

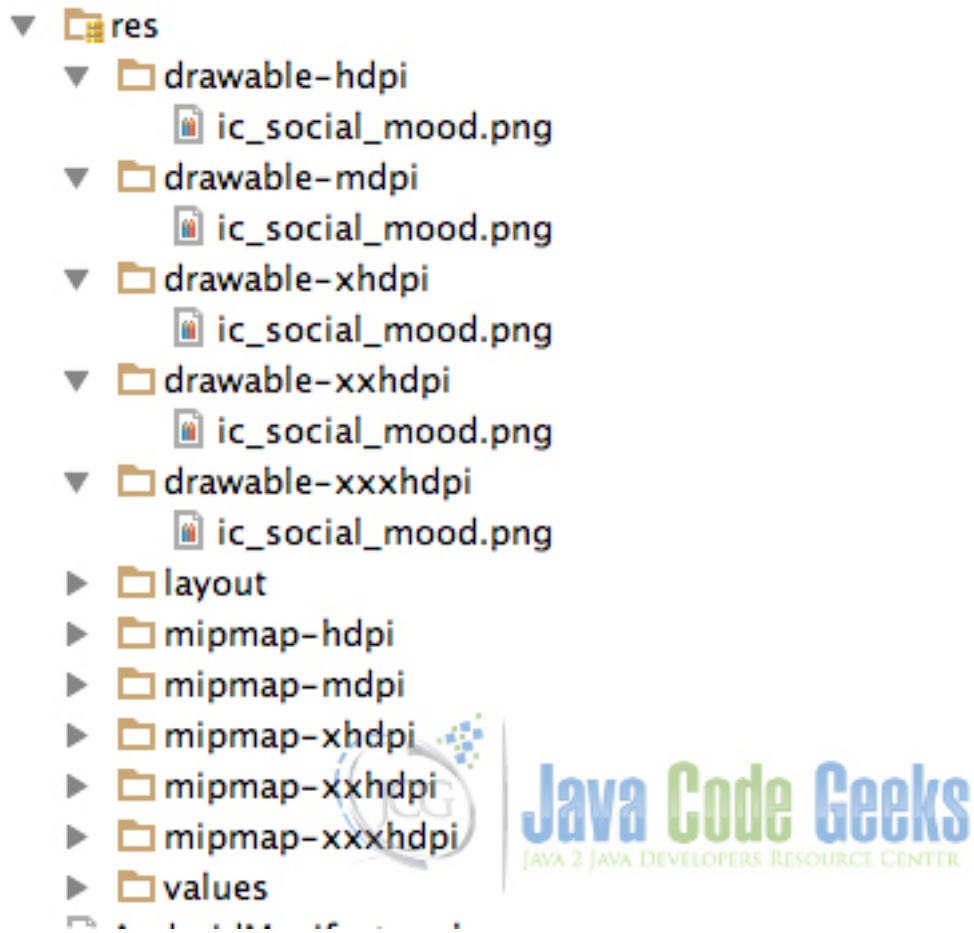

Figure 1.11: Add the drawables for every screen density.

In this way, we are going to have the right drawable dimension for every different screen density.

1.6.8 Build, compile and run

When we are ready, we build our application by pressing the play button in our AndroidStudio main toolbar.

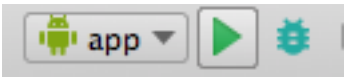

Figure 1.12: Compile and run.

After we build, compile and run our project, the main FirstAndroidApplication application should look like this:

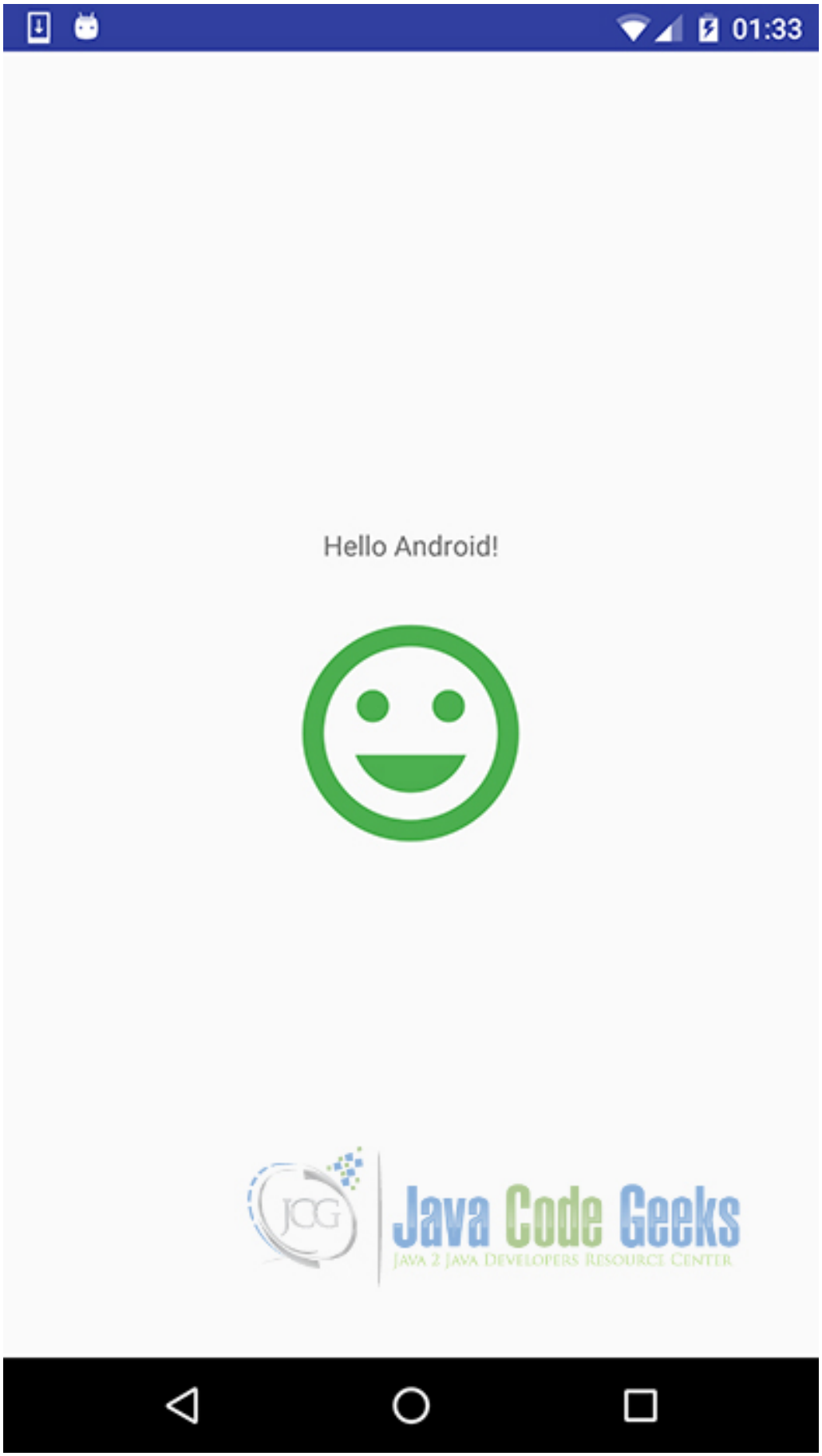

Figure 1.13: This is our FirstAndroidApplication.

1.7 Download the Android Studio Project

This was an example of first Android application.

Download You can download the full source code of this example here: **FirstAndroidApplication**

1.8 How to continue?

Here is a list of basic Android tutorials, that you can follow in order to make the first basic steps in the Android World:

Android Layouts and Views

- Android FrameLayout Example
- Android LinearLayout Example
- Android ImageView Example
- Android TextView Example
- Android Button Example

Android Click and Drag Listeners

- Android OnClickListener Example
- Android Drag and Drop Example

Android Styles and UI Elements

- Android Styles and Themes Example
- Android Toast Example
- Android Toolbar Example

Android Activities

- Android Activity Transition Example

Android Development

- Building Android Applications with Gradle
- Android Project migration from Eclipse to Android Studio

Of course the most accurate and complete guide, is the official Android Developers Guide, that covers every aspect of the Android Development:

- Official Android Developers Guide

Chapter 2

Android Project migration from Eclipse to Android Studio

Android Studio is the official IDE for Android development, and with a single download includes everything you need to begin developing Android apps.

This example describes the differences between Eclipse ADT and Android Studio, including project structure, build system, and application packaging, and will help you migrate your Android Eclipse project to Android Studio as your new development environment.

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Android Studio 1.3.2
- Eclipse 4.2 Juno
- Android SDK

Let's start with a slice of Android Studio theory...

2.1 Why to use Android Studio over Eclipse ADT?

Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine
- Rich layout editor with support for drag and drop theme editing
- lint tools to catch performance, usability, version compatibility, and other problem
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine
- Official Google Support and usual updates that need no migration

2.2 Android Studio new project structure

Eclipse provides workspaces as a common area for grouping related projects, configurations, and settings. In Android Studio, each instance of Android Studio contains a top-level project with one or more app modules. Each app module folder contains the equivalent to an Eclipse project, the complete source sets for that module, including `src/main` and `src/androidTest` directories, resources, build file, and the Android manifest.

In general, to update and build your app you modify the files under each module's `src/main` directory for source code updates, the `gradle.build` file for build specification, and the files under `src/androidTest` directory for test case creation. Also due to the structural differences between Android Studio projects vs Eclipse ADT projects, they cannot co-exist. Here is a table of the main differences:

Eclipse ADT	Android Studio
Workspace	Project
Project	Module
Project-specific JRE	Module JDK
Classpath variable	Path variable
Project dependency	Module dependency
Library Module	Library
`AndroidManifest.xml`	`app/src/main/AndroidManifest.xml`
`assets/`	`app/src/main/assets`
`res/`	`app/src/main/res/`
`src/`	`app/src/main/java/`
`tests/src/`	`app/src/androidTest/java/`

Figure 2.1: Eclipse - Android Studio Comparison

2.3 Gradle and build.gradle

Gradle is a build and automation tool, that can automate our building, testing, deploying tasks and many more. Gradle is the next generation build system for Java technologies that includes some advantages from older tools like Ant or Maven systems. Android Studio uses the power of Gradle, in order to provide all the above advantages, such as build variants and multiple apk file generation.

Android Studio projects contain a top-level build file and a build file for each module. The build files are called **build.gradle**, and they are plain text files that use Groovy syntax to configure the build with the elements provided by the Android plugin for Gradle. In most cases, you only need to edit the build files at the module level.

It looks like this:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 19
    buildToolsVersion "19.0.0"

    defaultConfig {
        minSdkVersion 8
        targetSdkVersion 19
```

```
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile project(":lib")
    compile 'com.android.support:appcompat-v7:19.0.1'
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

2.4 Simple Eclipse ADT project migration to Android Studio

Here, we have an example of this Eclipse ADT project migration to Android Studio. In this example, we are going to migrate the eclipse project that we created in this example: Android Google Maps v2 Tutorial. This is a wonderful example of how we are going to migrate a simple application project, that has a java class package and a Google Play Services library dependency. So, we are going to take this code, import it and compile it under Gradle system, and run it.

Open Android Studio and choose "Start a new Android Studio Project" in the welcome screen.

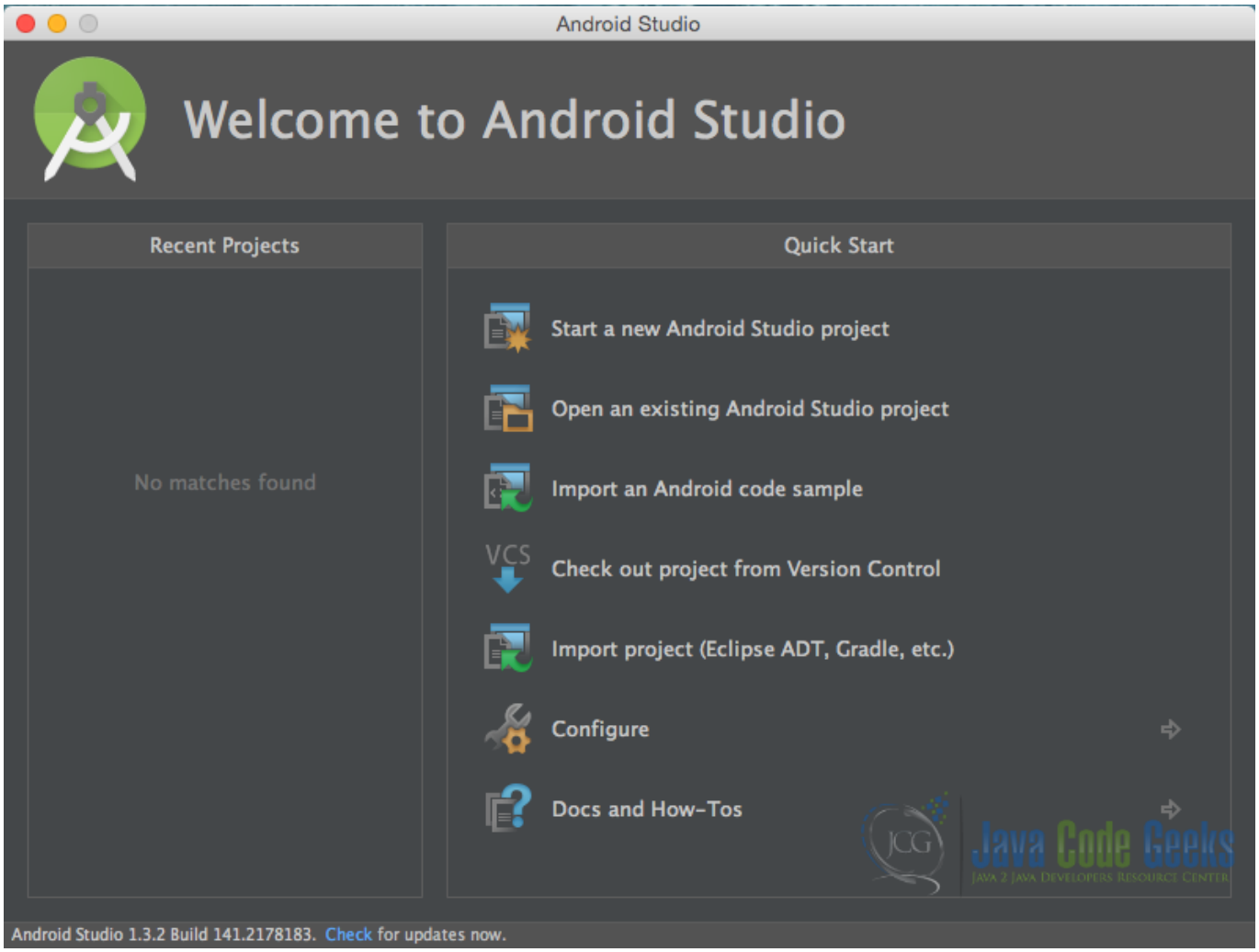

Figure 2.2: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

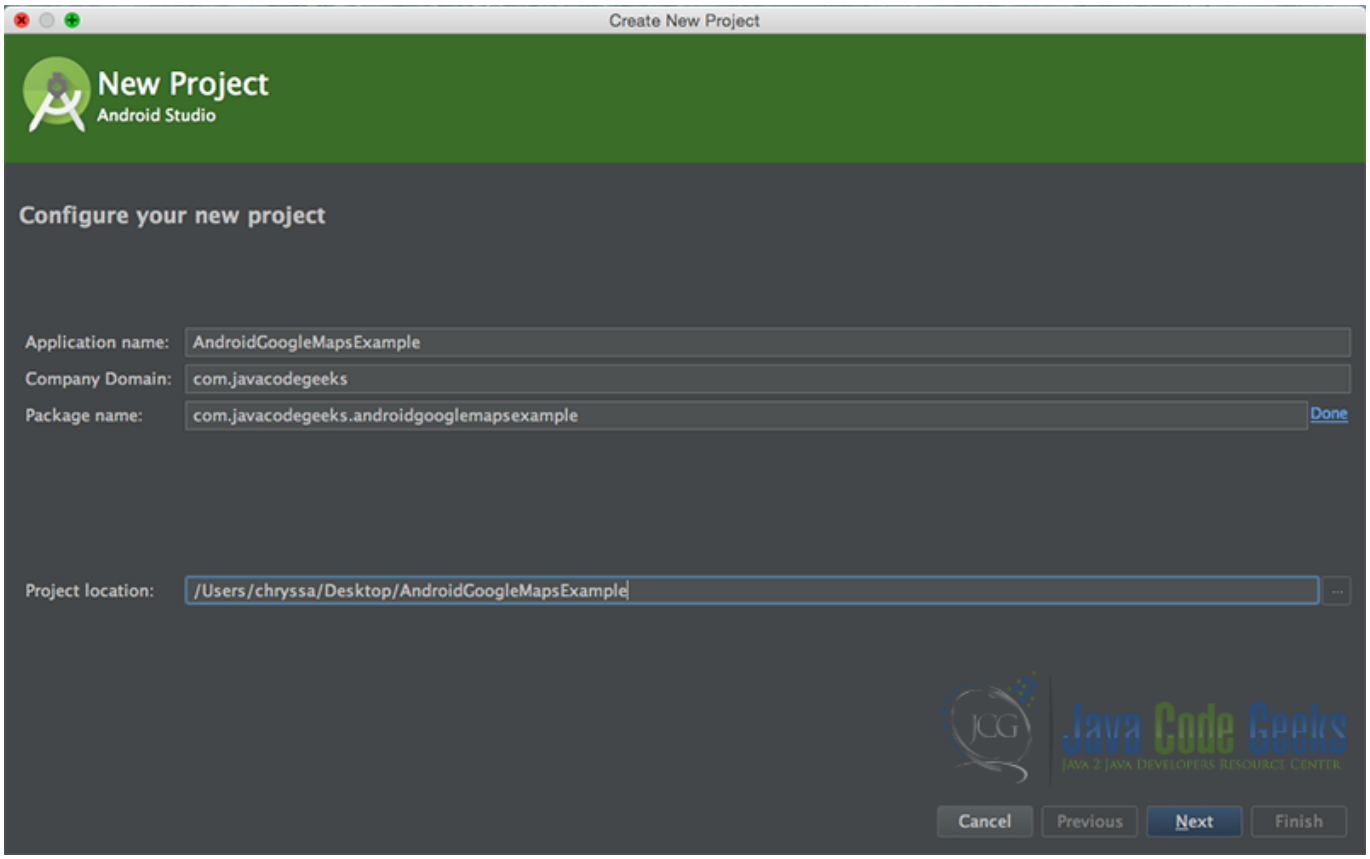

Figure 2.3: "Configure your new project" screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

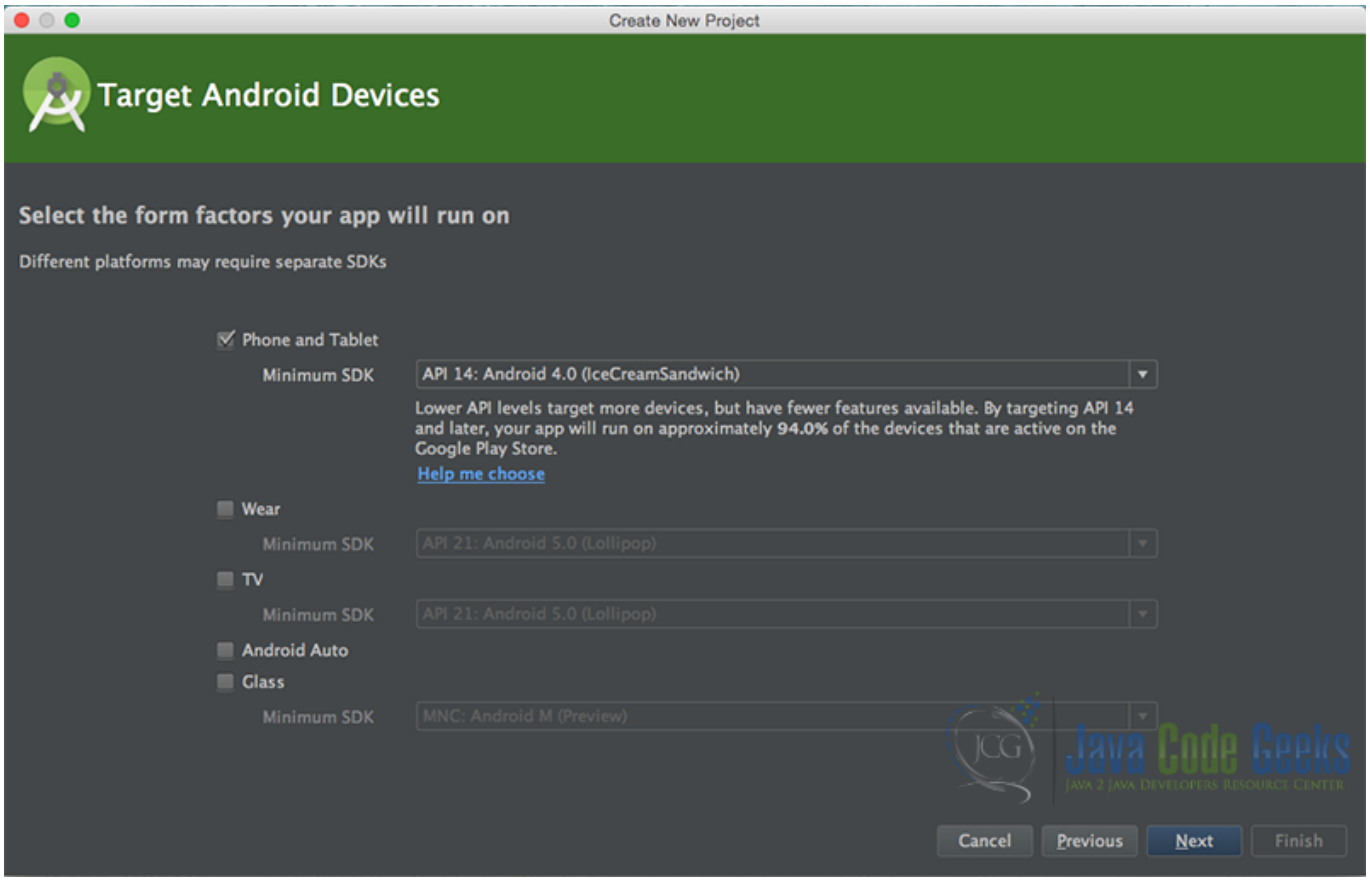

Figure 2.4: "Target Android Devices" screen.

In the next window you should choose to "Add an activity to Mobile". In our example, we will choose to create a project with no activity, because we will migrate our Activities for the eclipse formed project. So, choose: "Add no activity".

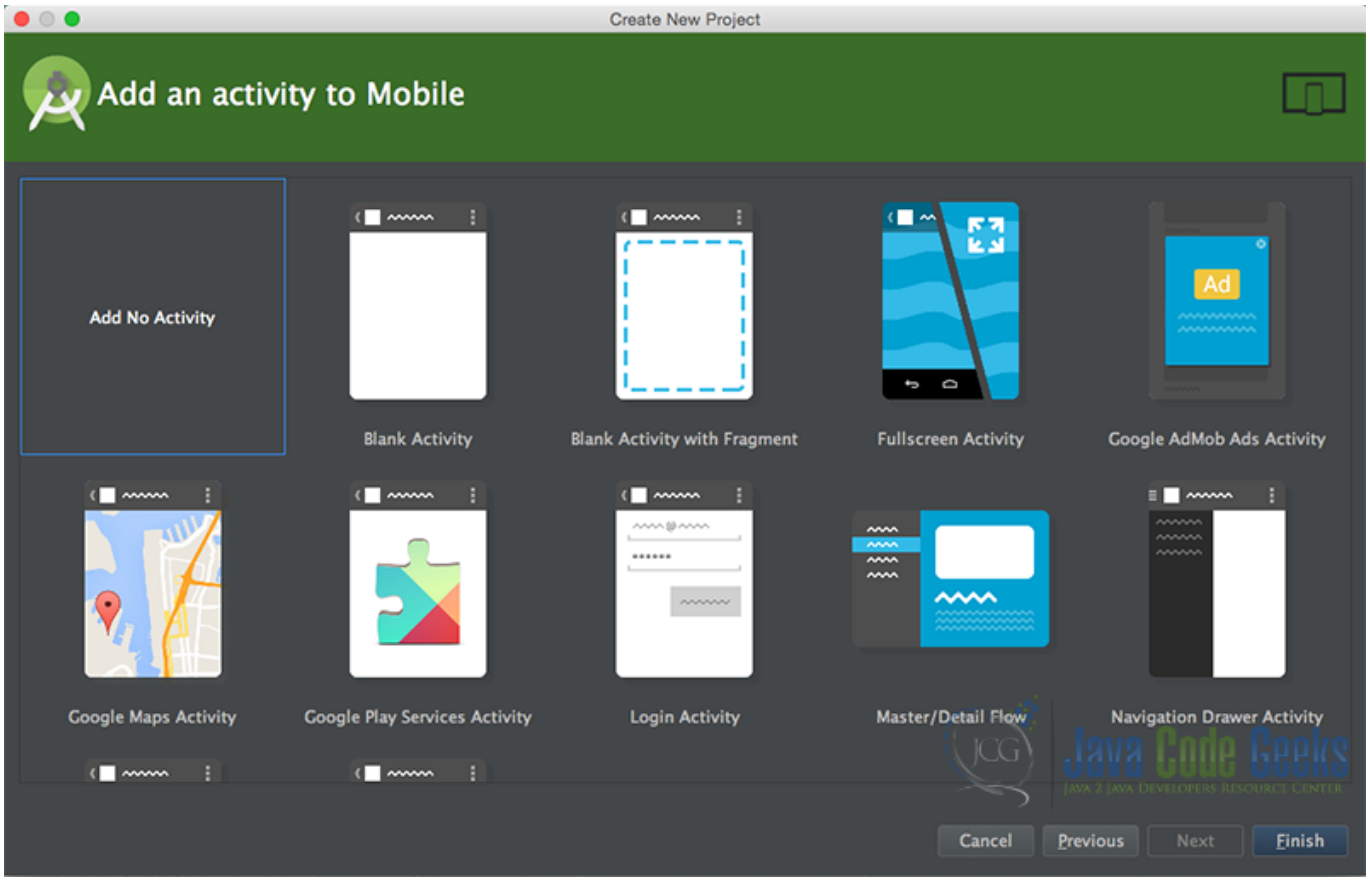

Figure 2.5: Add an activity to Mobile. Choose: Add no activity.

Now, our project has just been created. This is how it looks like in the "Android" project view:

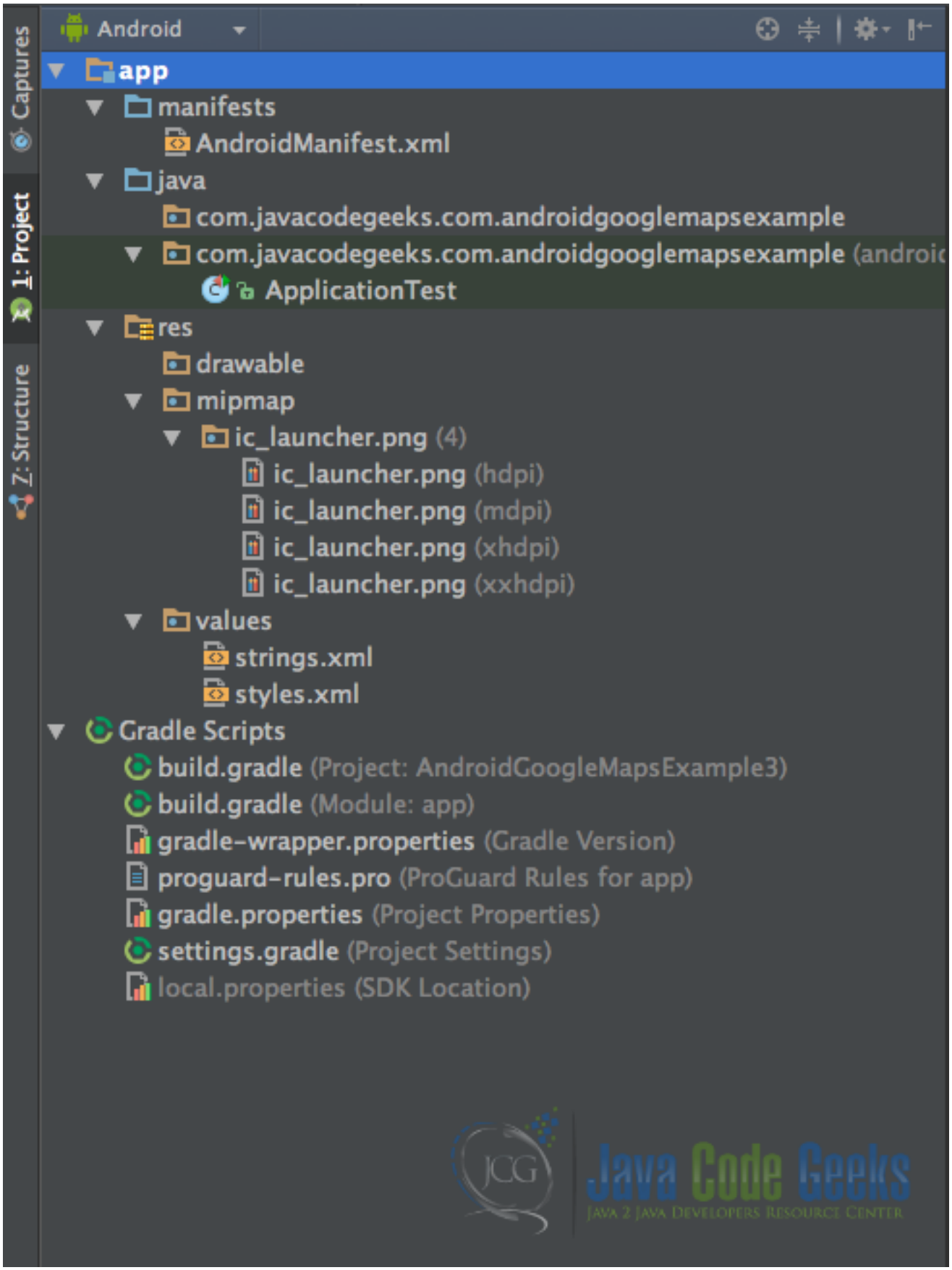

Figure 2.6: A new Android Studio project has just been created. This is how it looks like.

2.5 Java code and resources migration

As we discussed above, there are some pretty significant changes between the project structures between Eclipse ADT and Android Projects. The biggest is that both Java classes and the Android resources folder, are under `app/src/main/` directory. We are going to copy our Java classes alone, inside the `app/java/com.javacodegeeks.androidgooglemapsexample` folder, as we see it in Android package view.

After this, we are going to copy also our eclipse resources folders under `app/res/` folder, as we see it in Android package view. If this suggest to overwrite some files and folders, we do it cautiously. We should now have something like this:

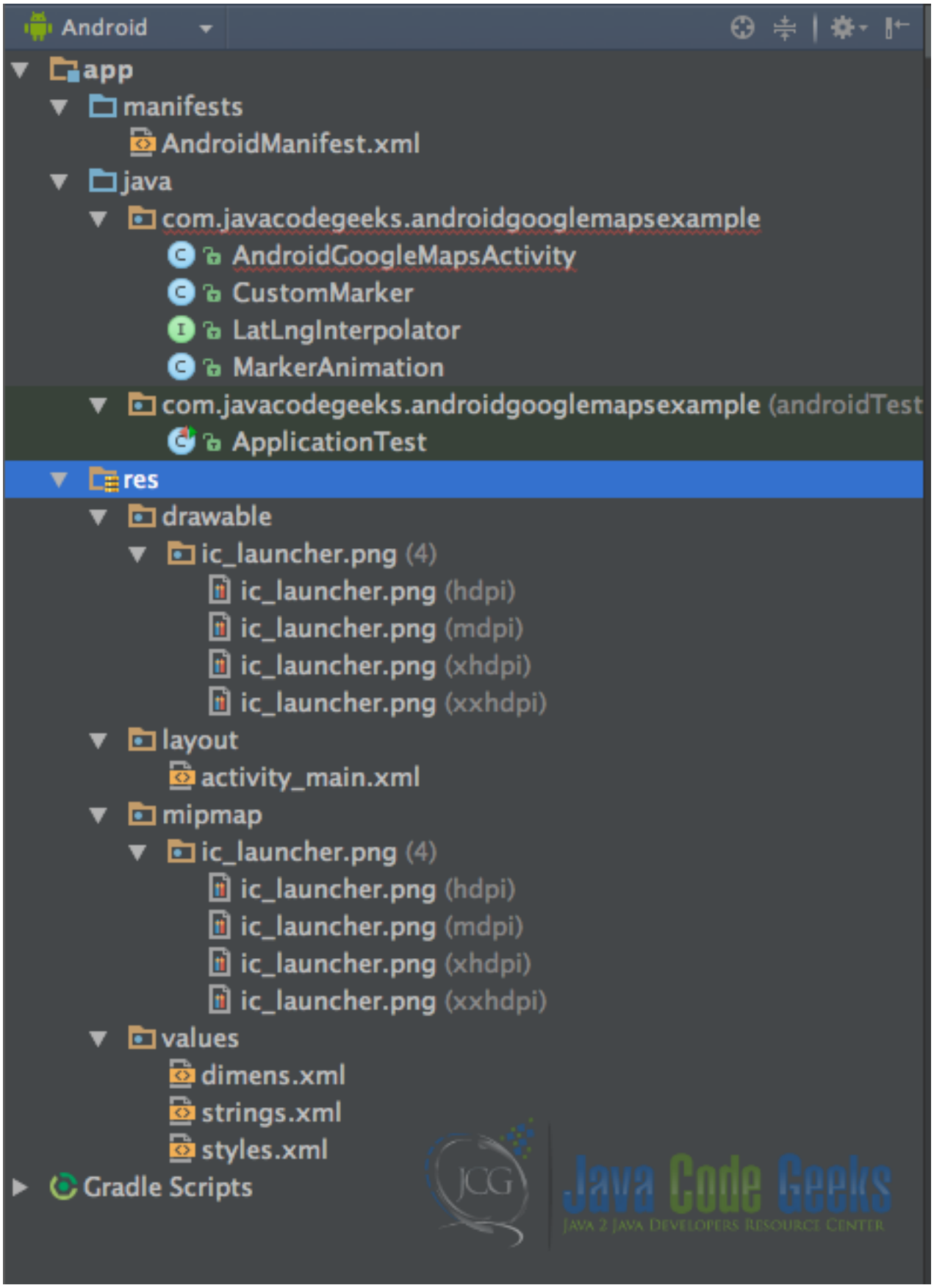

Figure 2.7: This is how our projects looks like, after we have also moved inside our Java and resource folders.

2.6 AndroidManifest.xml and build.gradle file

Then, we move on copying our AndroidManifest.xml file. In Android Studio project structure, we can find our manifest files inside the `app/manifests` folder. We overwrite the Android Studio project AndroidManifest.xml with our eclipse project manifest xml.

We should now have something like this:

AndroidGoogleMapsActivity.java | AndroidManifest.xml | app

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.androidgooglemapsexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="19" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <!--
     The following two permissions are not required to use
     Google Maps Android API v2, but are recommended.
    -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".AndroidGoogleMapsActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="
    </application>

</manifest>
```

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Figure 2.8: This is our AndroidManifest.xml. This is located in app manifests folder

Finally, we have our build.gradle file, that we should be very careful in its configurations.

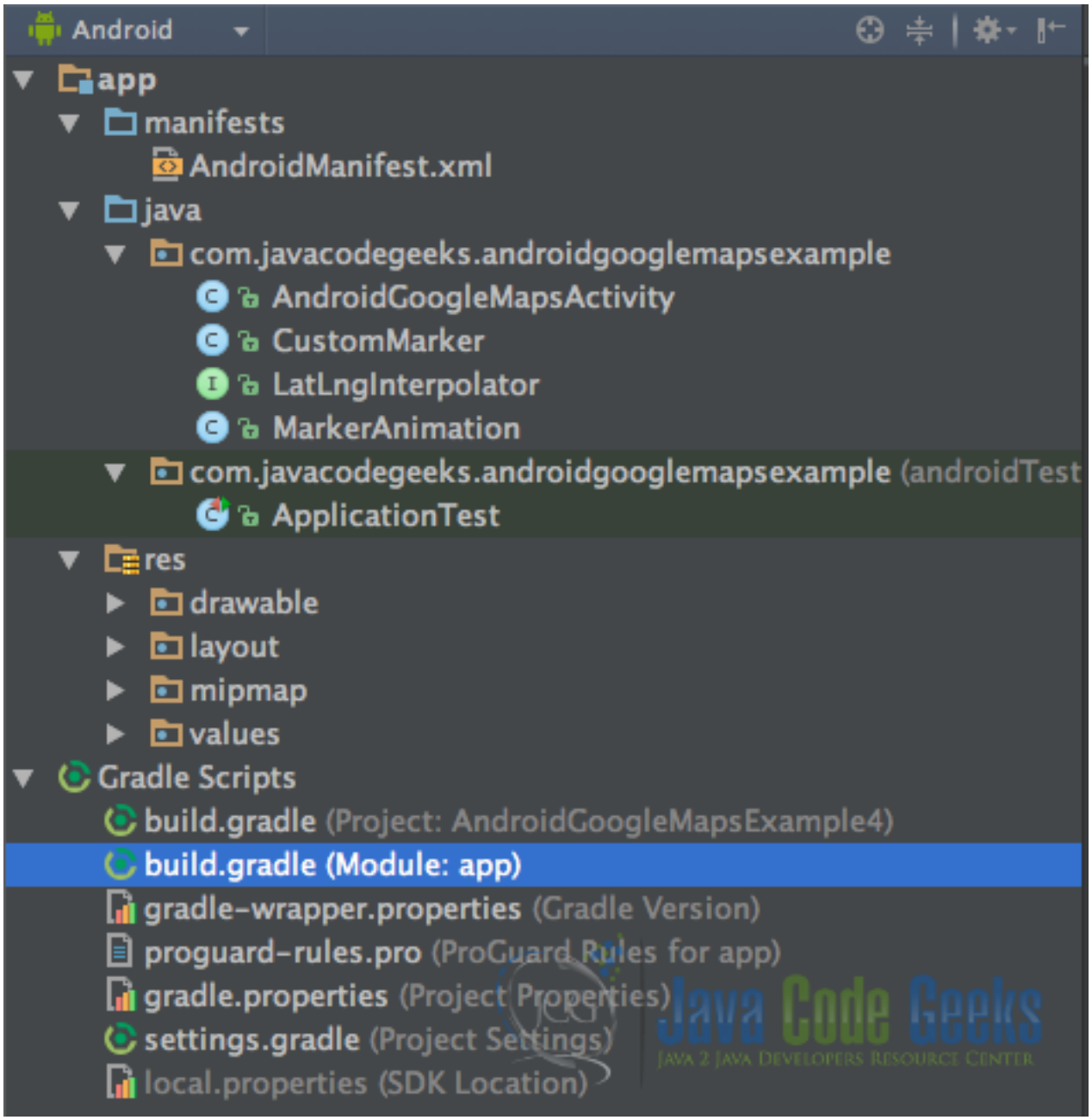

Figure 2.9: Here is our build.gradle file.

We write something like this:

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "23.0.1"

    defaultConfig {
        applicationId "com.javacodegeeks.androidgooglemapsexample"
        minSdkVersion 14
        targetSdkVersion 22
```

```
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.google.android.gms:play-services:8.1.0'
}
```

With `compile fileTree(dir:'libs', include:['*.jar'])` we add in our Gradle configuration, any external library we might have added in the `app/libs` project folder. And with the next line `compile 'com.google.android.gms:play-services:8.1.0'` we add in our Gradle configuration the public repository in which Google supports Google Play Services library with Gradle. In this way we have added Google Play Services library in our project. This library is going to be compiled and packaged in our application project!

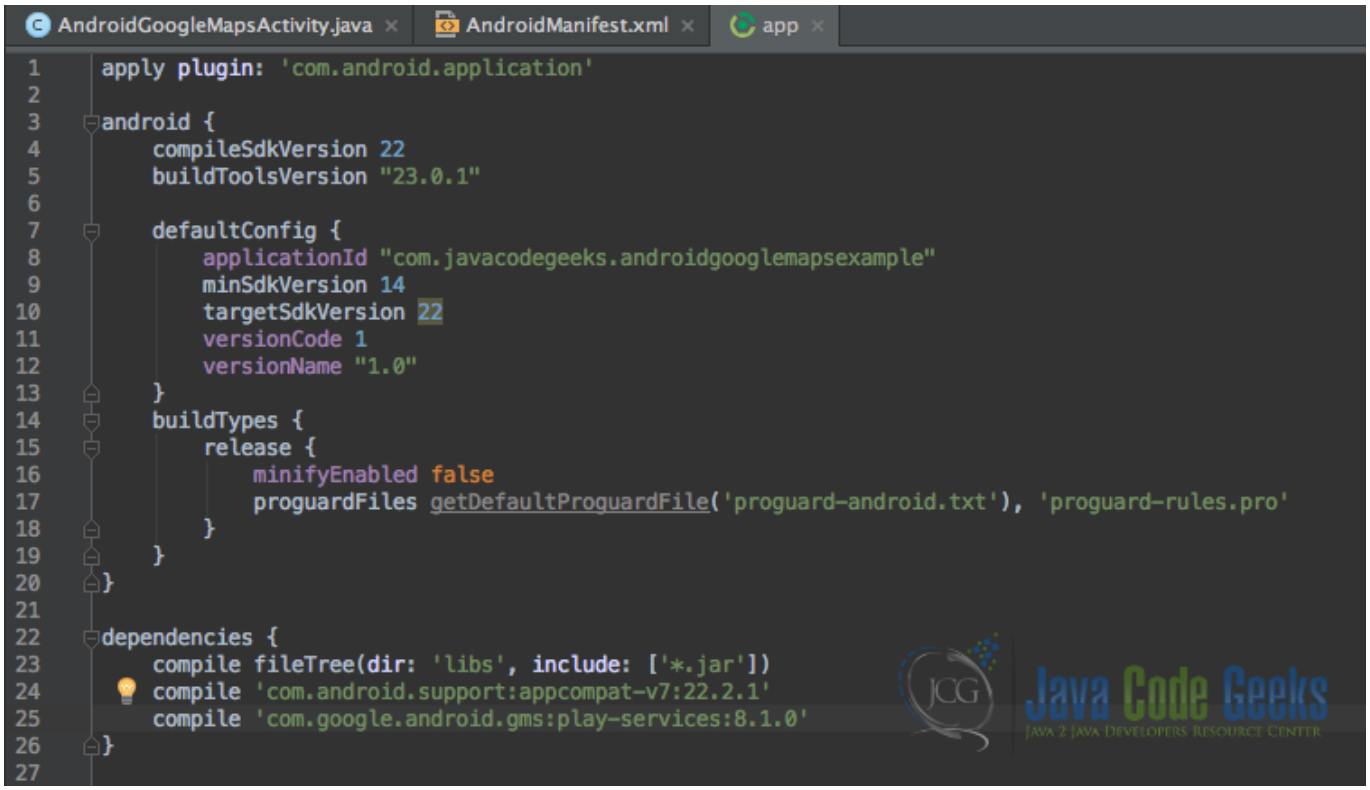

Figure 2.10: This is our build.gradle configuration file.

We, now, have to sync our project, and run this module, by pressing the "run" green button. If everything is the right place, and especially the application package names are the right ones, then we should see our project run.

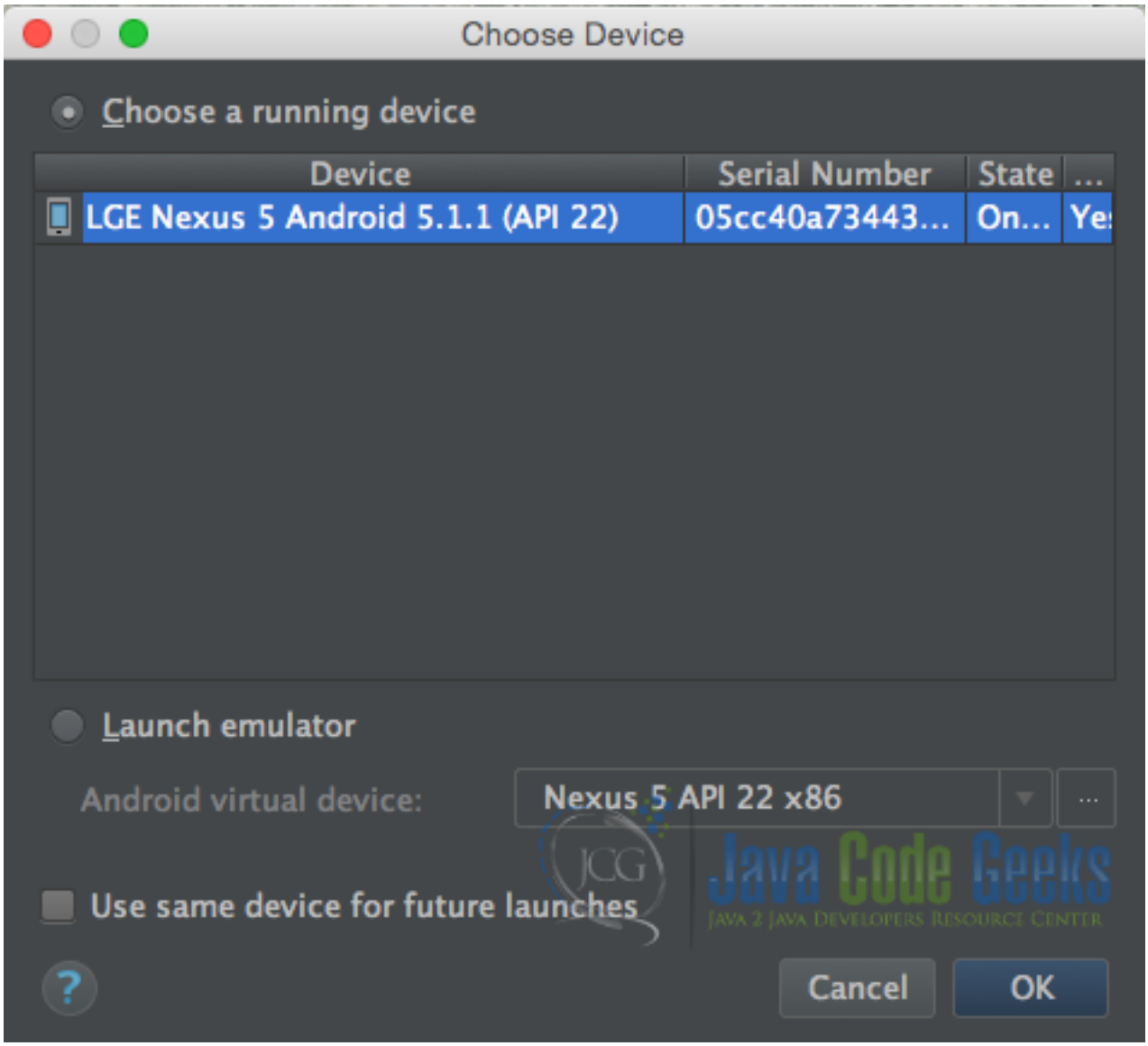

Figure 2.11: This is the running confirmation screen.

This was the Android Project migration from Eclipse to Android Studio example. This example was a theoretical one. From now on we are going to present examples in Android Studio IDE, as Google has stopped support on Eclipse ADT. Additionally, Android Studio is now the official Android IDE!

2.7 Download the Android Studio Project

This was an example of Android Google Maps v2 Tutorial migrated to Android Studio.

Download You can download the full source code of this example here: **AndroidGoogleMapsExampleAD**

Chapter 3

Android Google Maps v2 Tutorial

The great power of mobile applications software, is that it gives us the opportunity to develop many ideas that use hardware features, in order to complete simple everyday tasks. One idea that is implemented in a very good way in the mobile applications world, is the use of maps and the location based applications, that help us in multiple ways in our every day life. So, the use of maps is very important in various mobile applications.

Google provides via Google Play Services a library for using maps. This library is the second attempt of Google Maps and this version provides significant improvements to the older API version. So, in this example, we are going to work with Google Maps v2 and see how we will import a fully working map Fragment in an Activity, as well as, we are going to work with basic Markers and CameraPosition on Google Map.

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Eclipse 4.2 Juno
- Android SDK 4.4.2

Let's take a closer look:

3.1 Create a Google Maps API key

To begin with, we should create a new Google Maps API key and subscribe our application in order to grant access for our application to succesfully use Google Maps v2. In order to use this API, we will have to enter the Google API console portal. If you do not have an account, just create one or login with your current Google account.

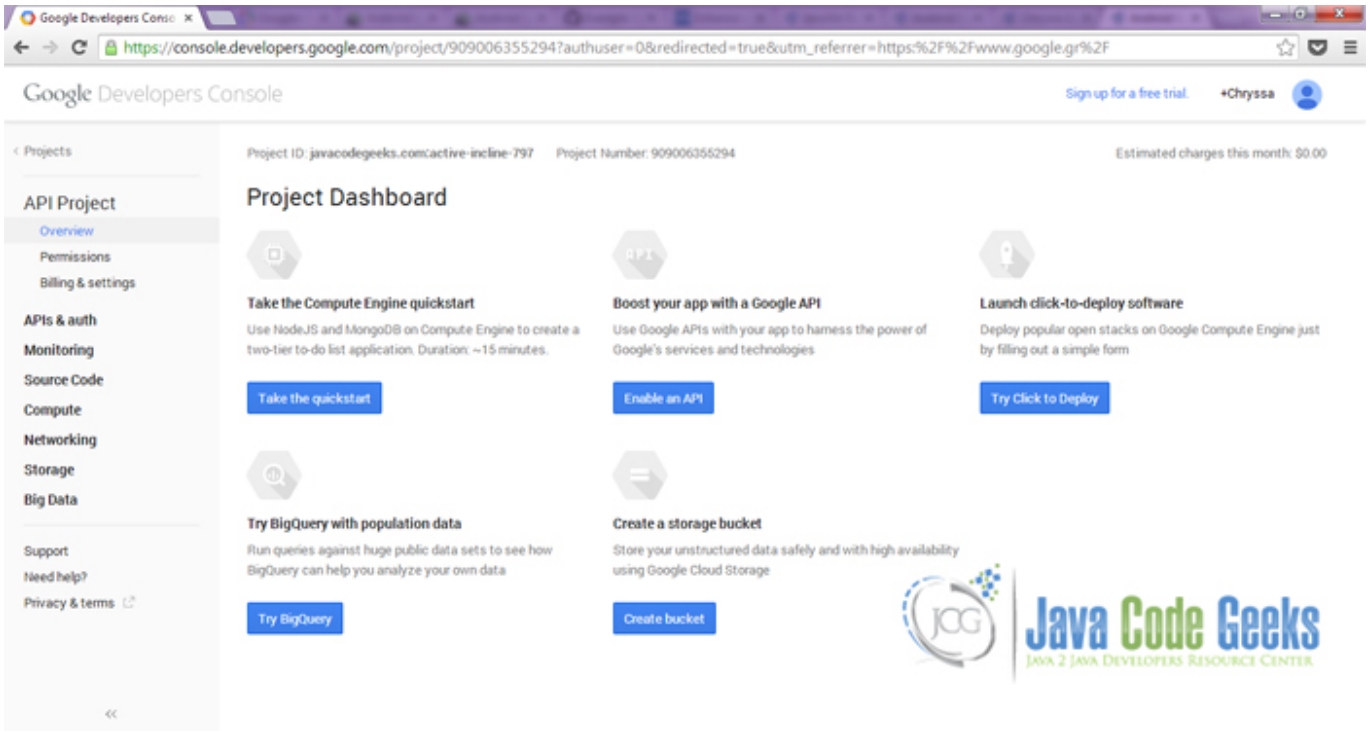

Figure 3.1: Enter Google API Console

After we enter the console, we are going to see the full list, of all the APIs that Google serves. Now we can see the enabled APIs that this console user has.

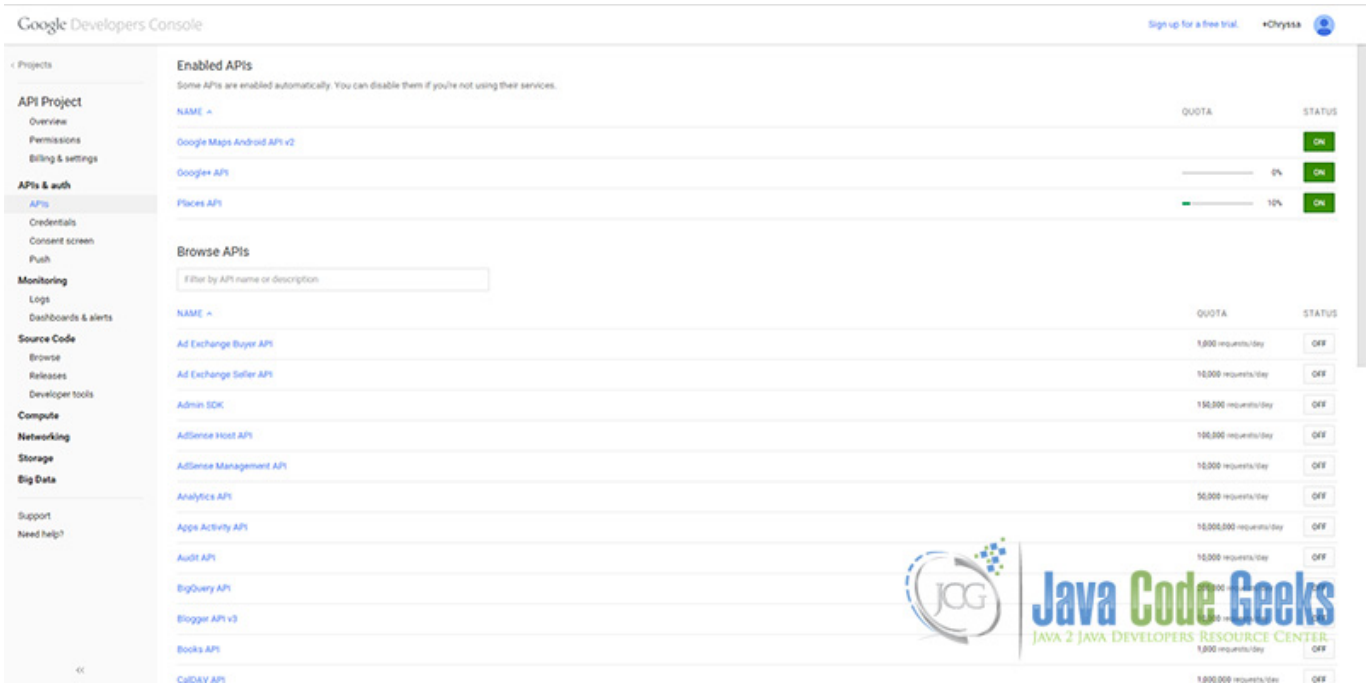

Figure 3.2: Figure 2. Go to APIs list

After we enter the console,we are going to see the full list, of all the APIs that Google serves. The API that we are going to use,

is the Google Maps v2 API. We click on the right service, and we turn it on. We have to accept the terms of Google Maps v2 API, by ticking the checkbox and by clicking the “Accept” button.

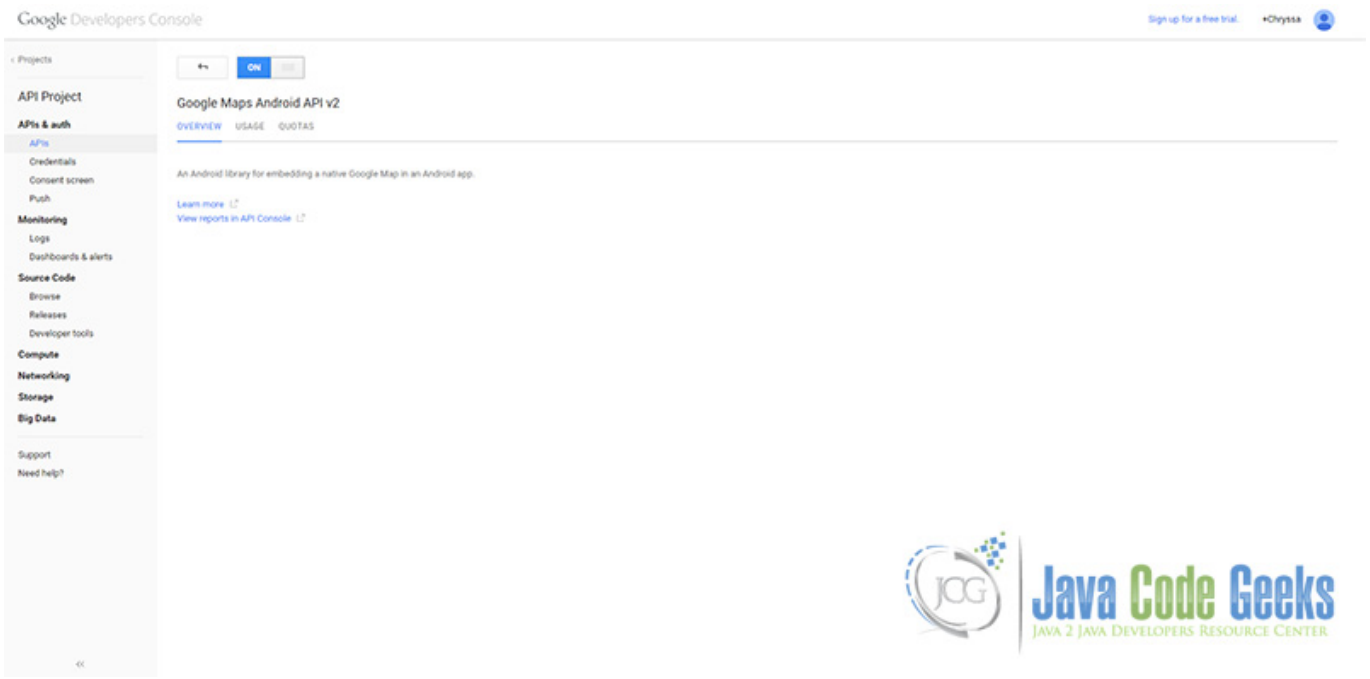

Figure 3.3: Enable Google Maps v2 API

We should see the Google Maps v2 API enabled.

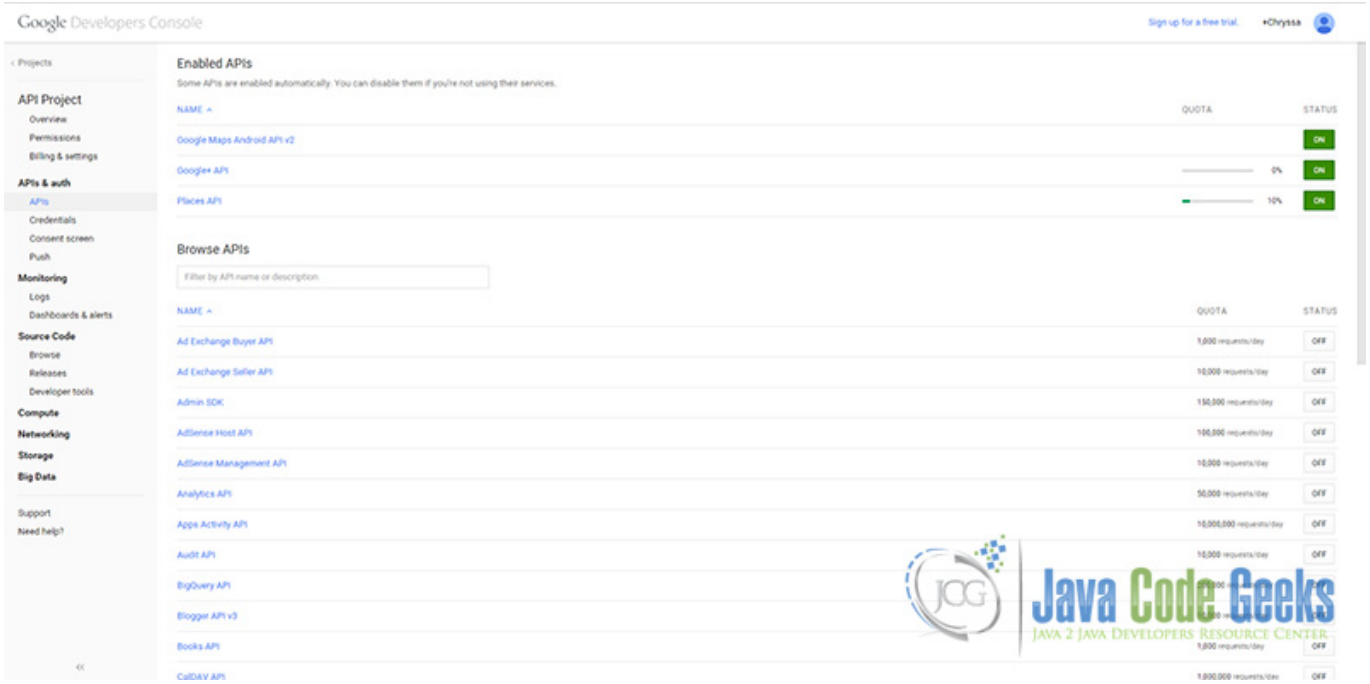

Figure 3.4: Google Maps v2 API enabled

We are going to create a Public API access credential, in order to take the right response from the Google Places API search call.

We only have to go to APIs&auth > Credentials tab.

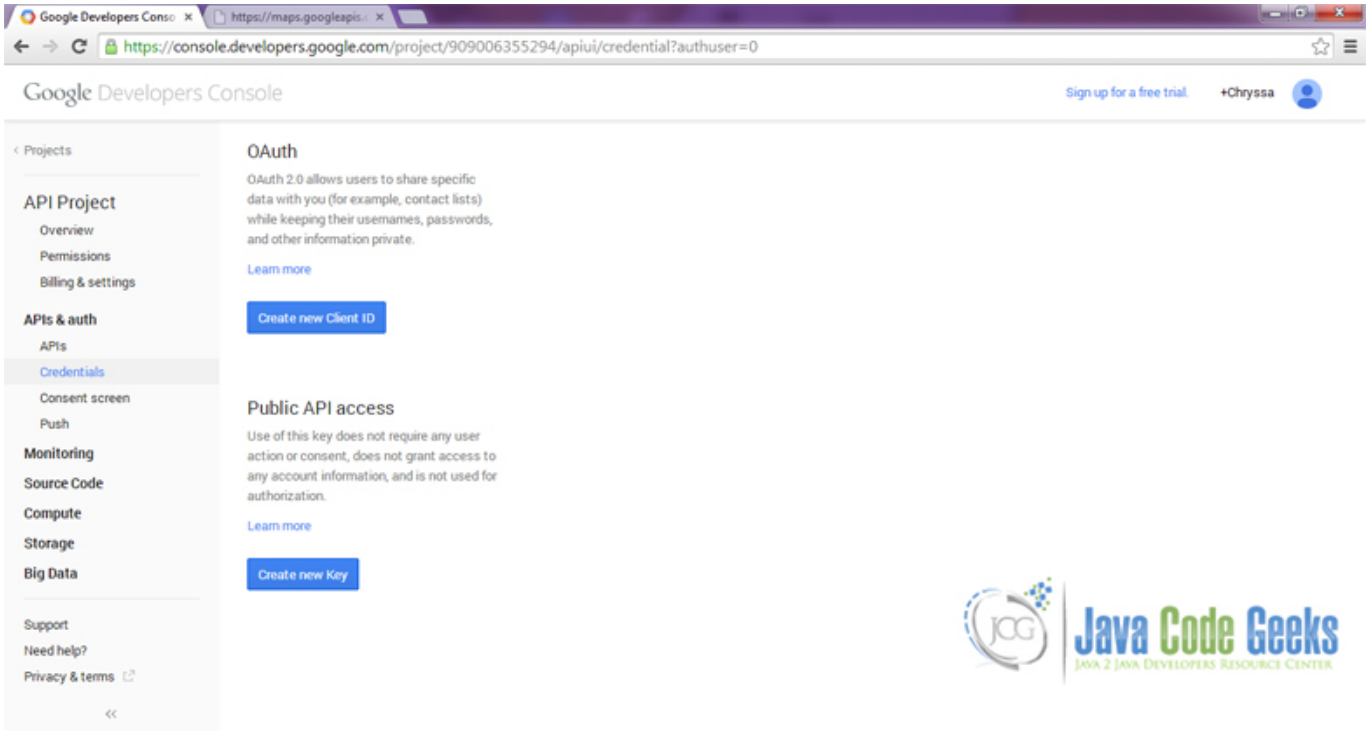

Figure 3.5: Credentials Key

We click the “Create new Key” button, and select the “Android Key” in order to make a Key that will both work from our Android Application and from our browser. We are going to do this, exclusively for our example scope.

If you want to publish an application to the Google Play Store, that uses any of these Google API calls, you should create an Android key, by using the SHA-1 key of your original keystore (not the debug one) of the application to be exported.

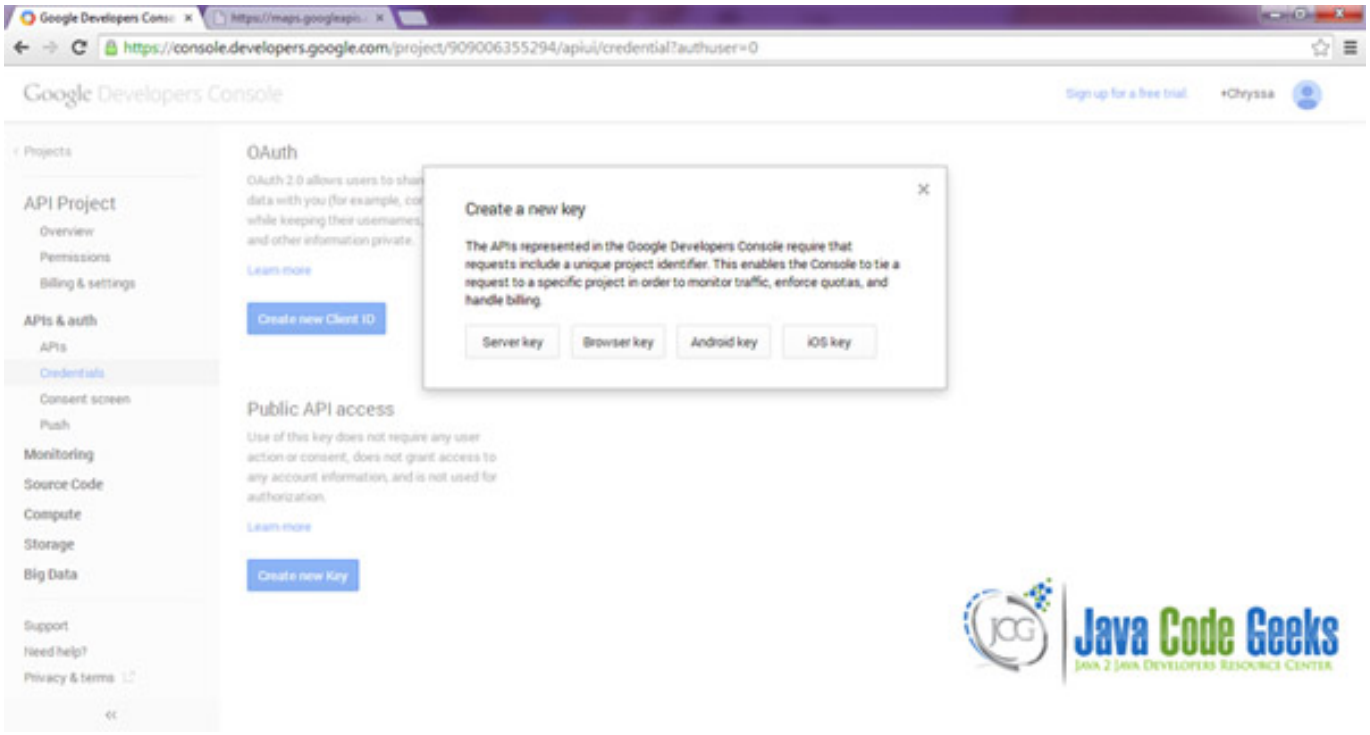

Figure 3.6: Create new Key

We are going to do this, exclusively for our application scope. So we have to provide our application's name and package. We have also to fill the SHA-1 cerificate fingerprint.

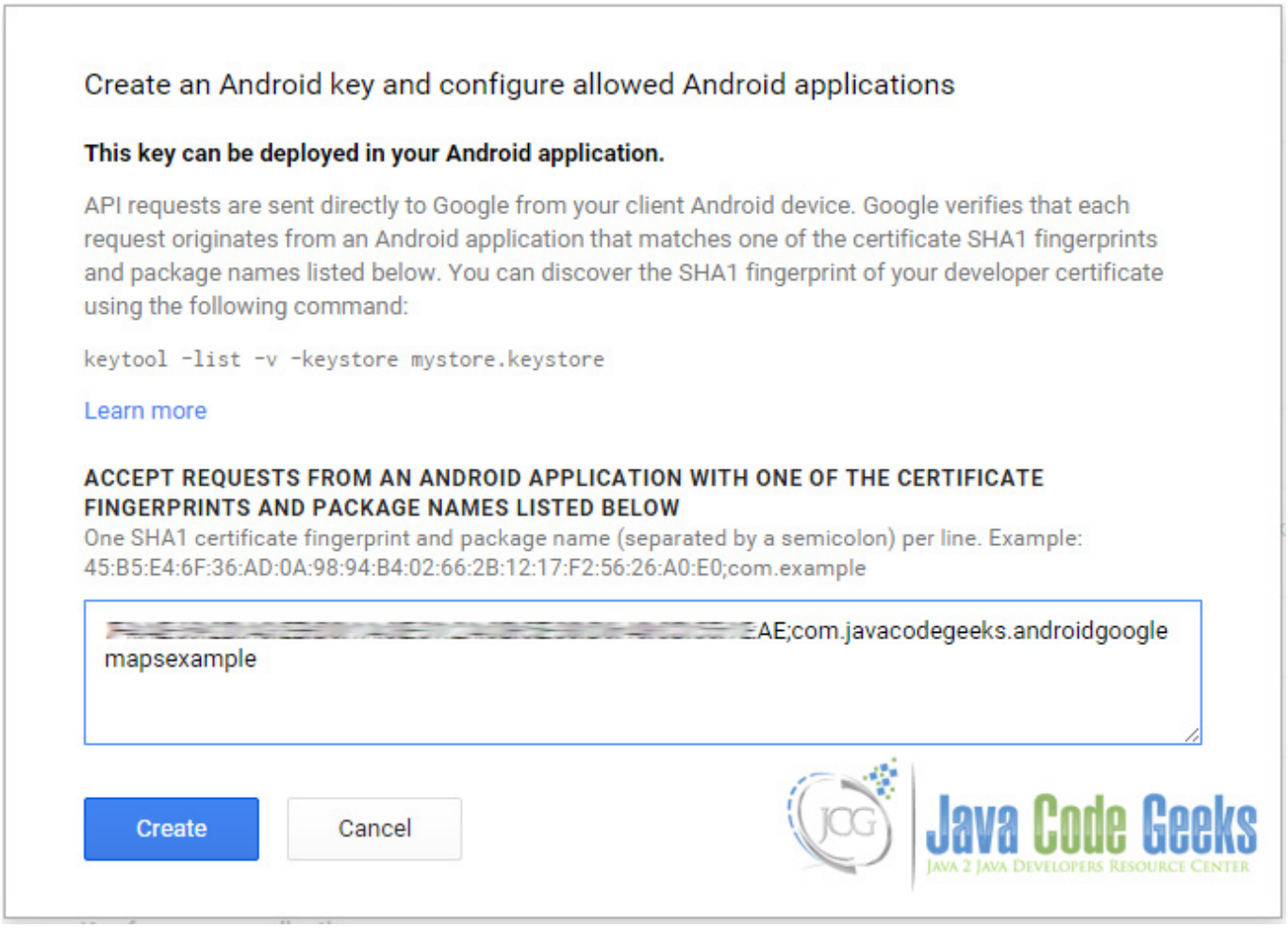

Figure 3.7: Configure the allowed IPs

If you are not familiar on how to get your SHA-1 cerificate fingerprint, the easiest way to reach this certificate, is inside your Eclipse IDE, from Window > Preferences > Android > Build.

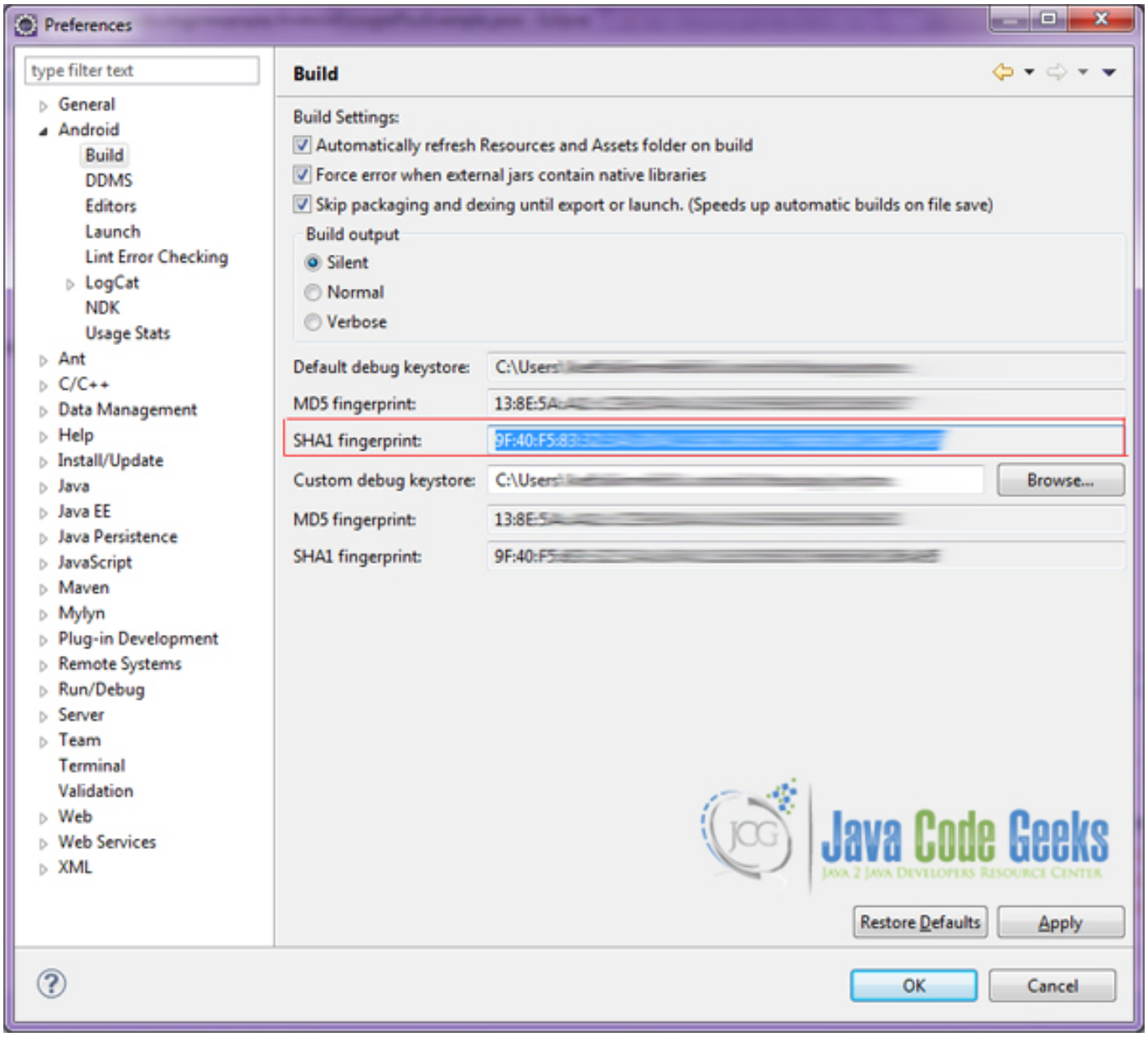

Figure 3.8: SHA-1 cerificate fingerprint on Eclipse

3.2 Create a New Android Application Project

Open Eclipse IDE and go to File → New → Project → Android Application Project.

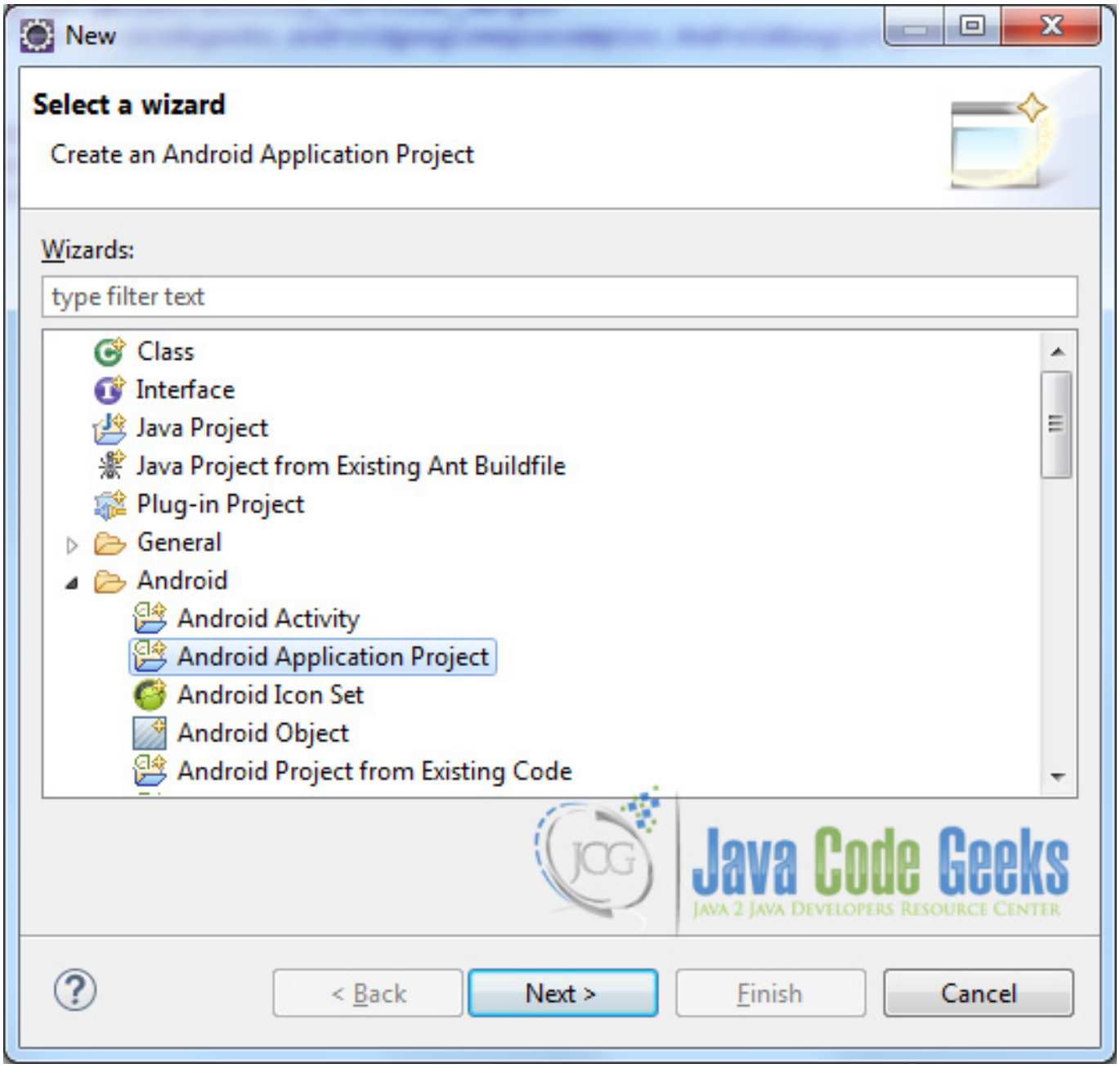

Figure 3.9: Create a new Android project

Specify the name of the application, the project and the package and then click Next.

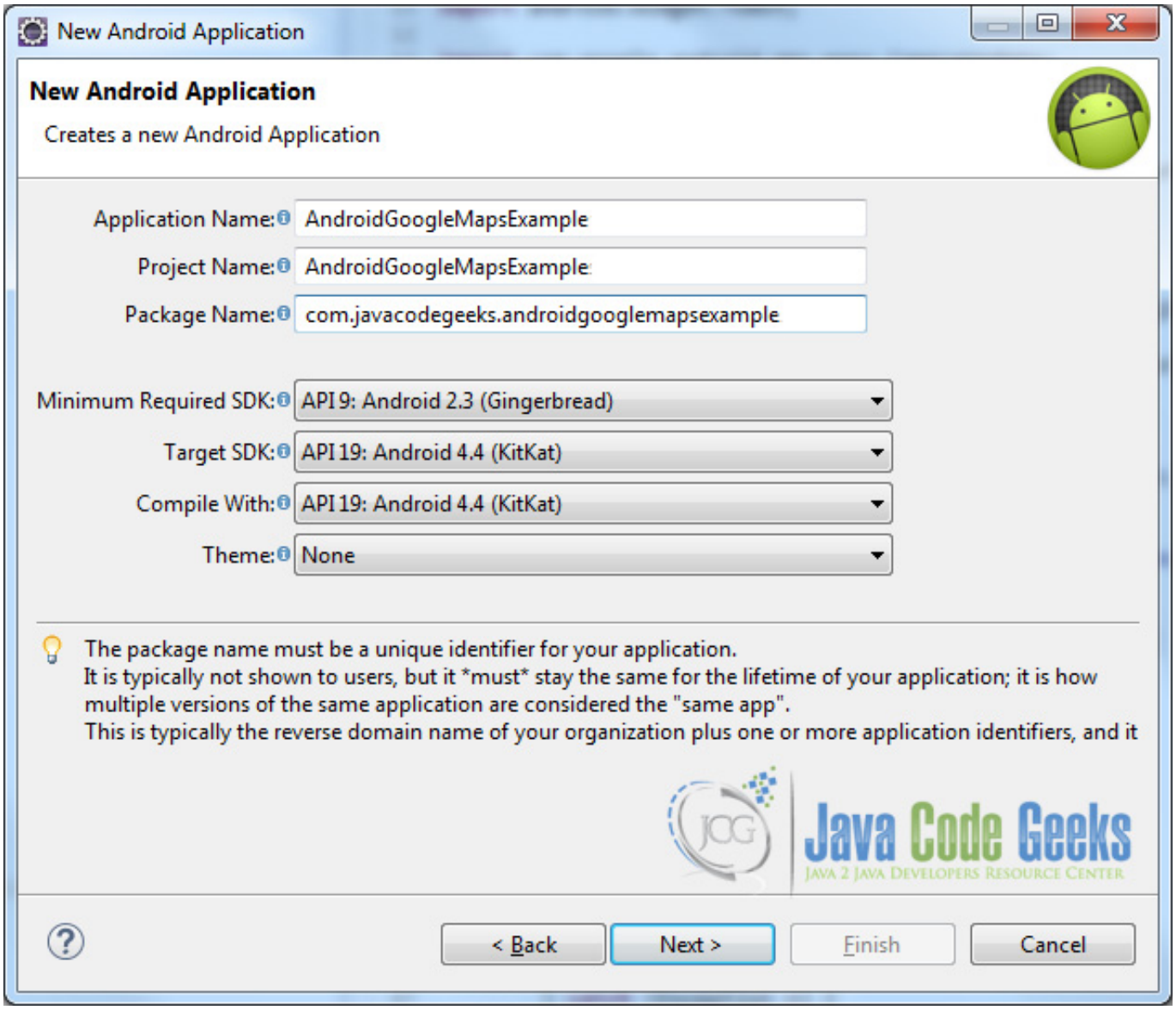

Figure 3.10: Create a new Android project name

In the next window, the “Create Activity” option should be checked. The new created activity will be the main activity of your project. Then press Next button.

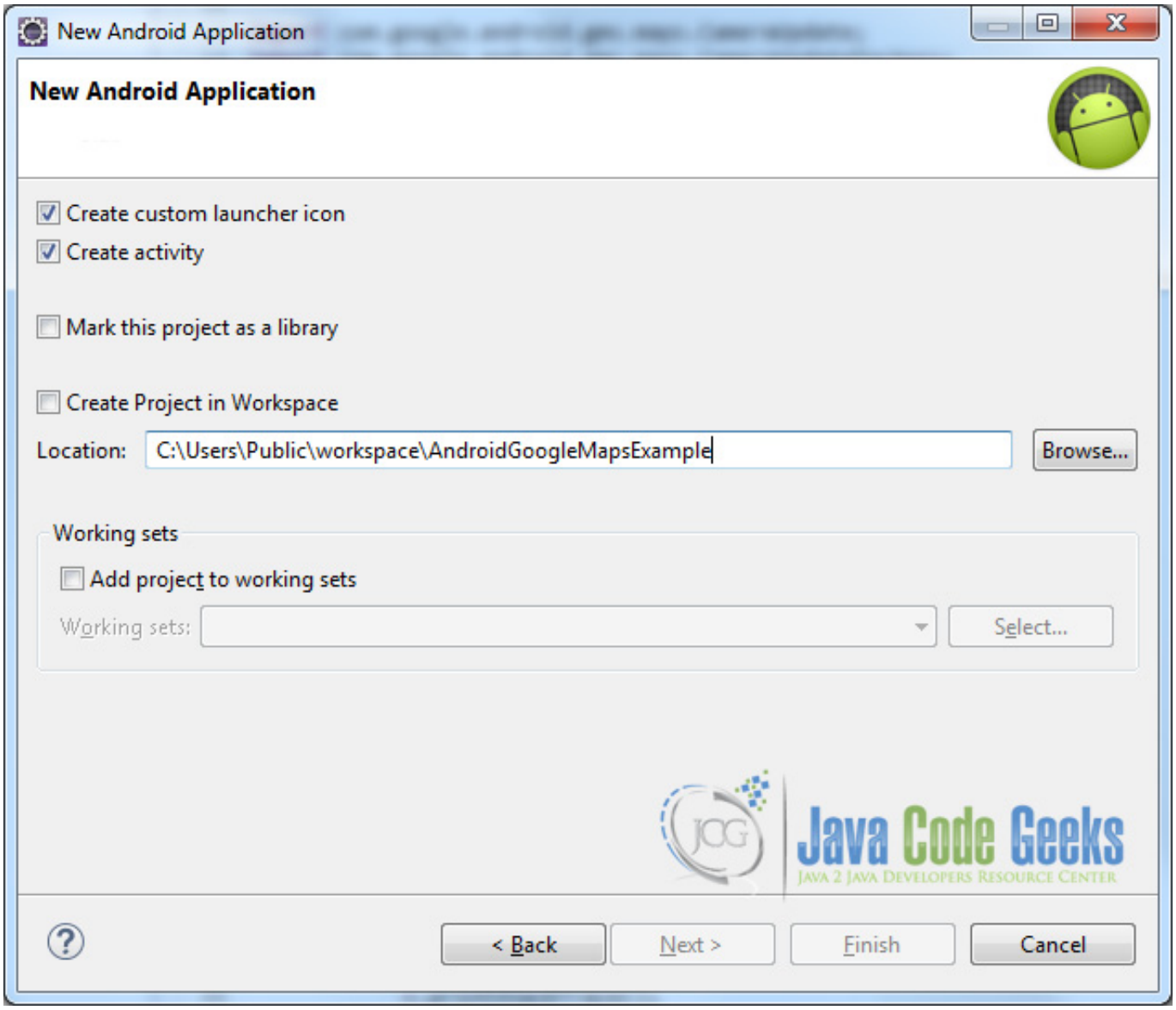

Figure 3.11: Configure the project

In “Configure Launcher Icon” window you should choose the icon you want to have in your app. We will use the default icon of android, so click Next.

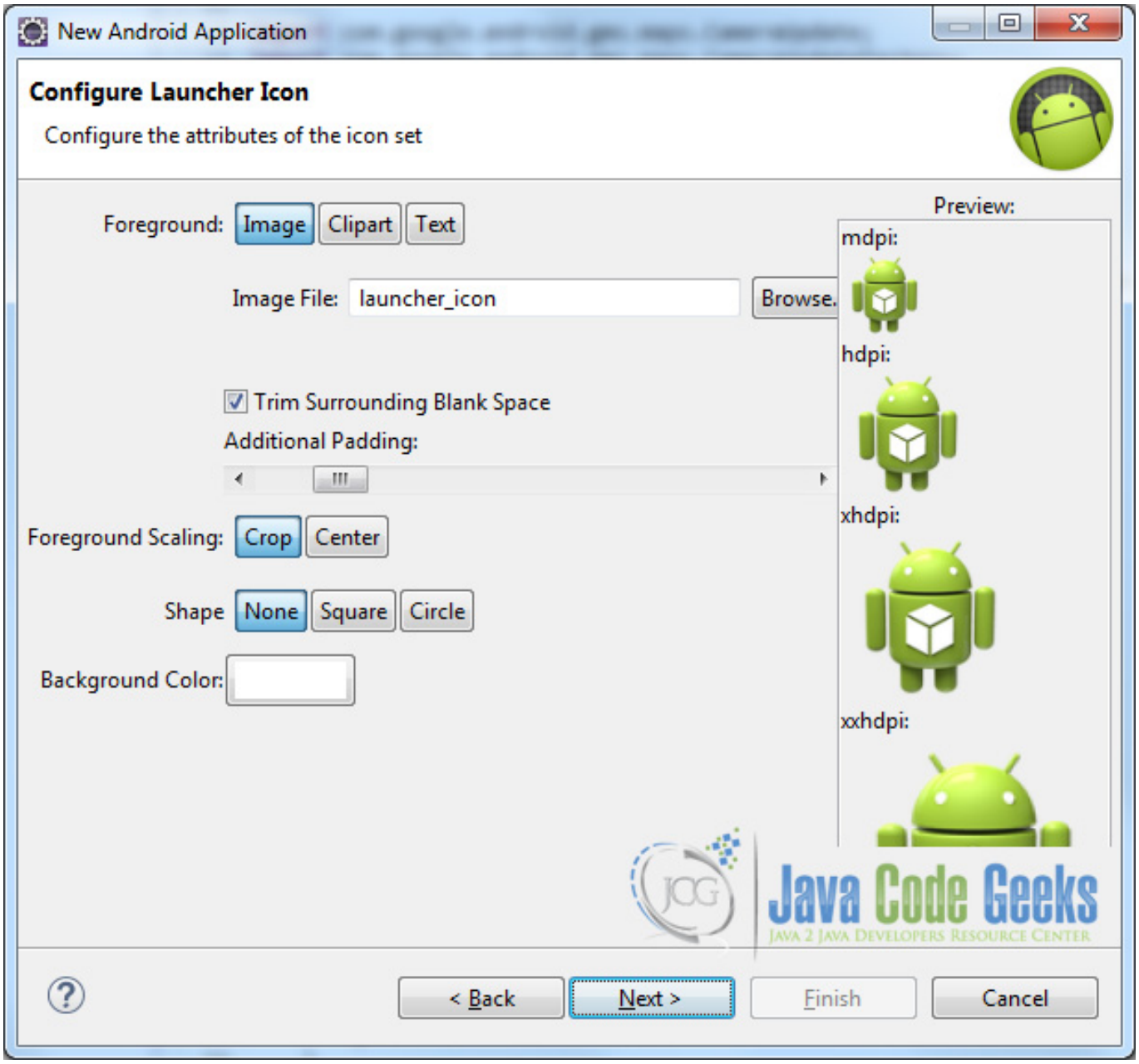

Figure 3.12: Configure the launcher icon

Select the “Blank Activity” option and press Next.

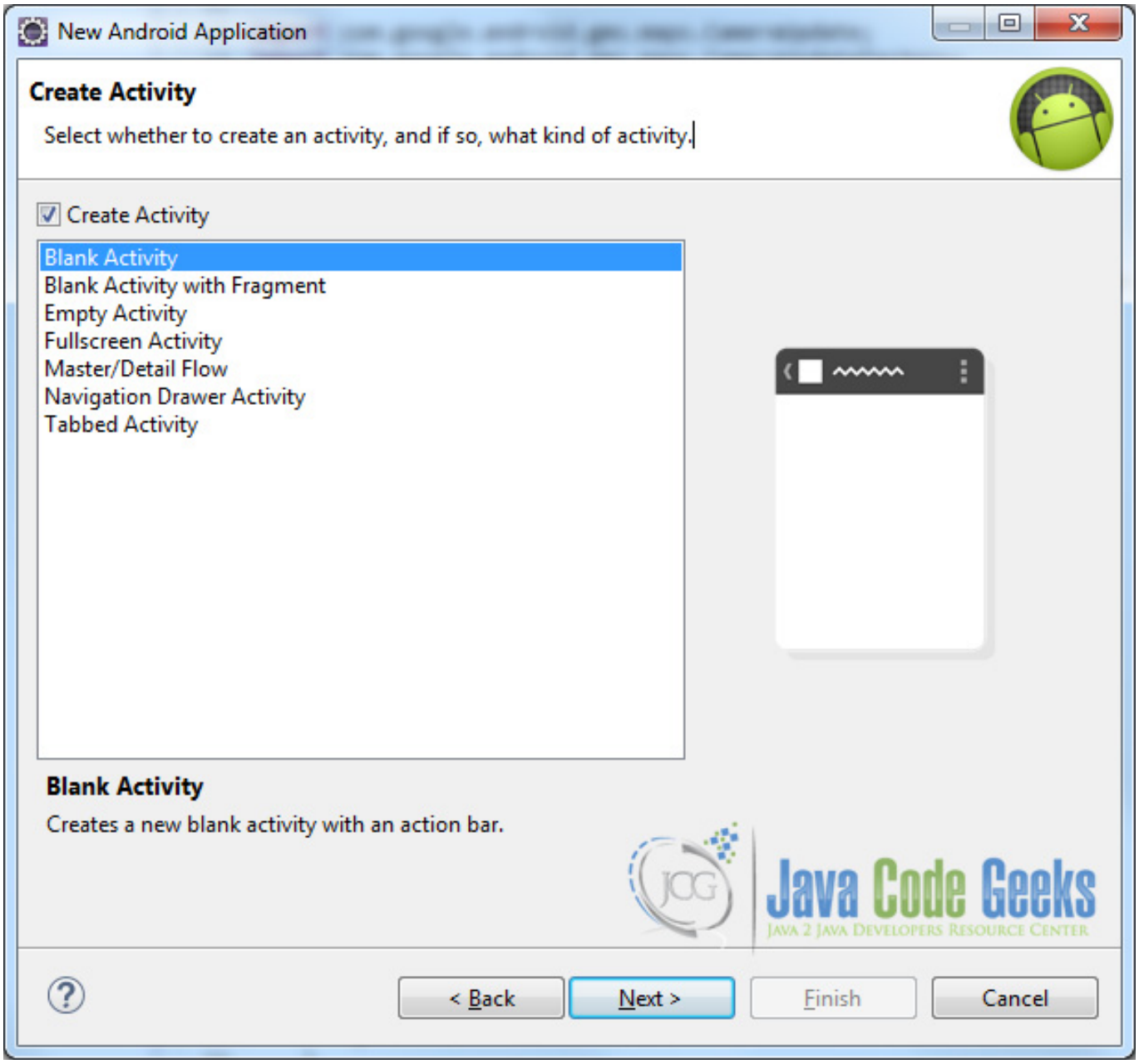

Figure 3.13: Create the activity and select its type

You have to specify a name for the new Activity and a name for the layout description of your app. The .xml file for the layout will automatically be created in the res/layout folder. It will also be created a fragment layout xml, that we are not going to use in this project and you can remove it if you want. Then press Finish.

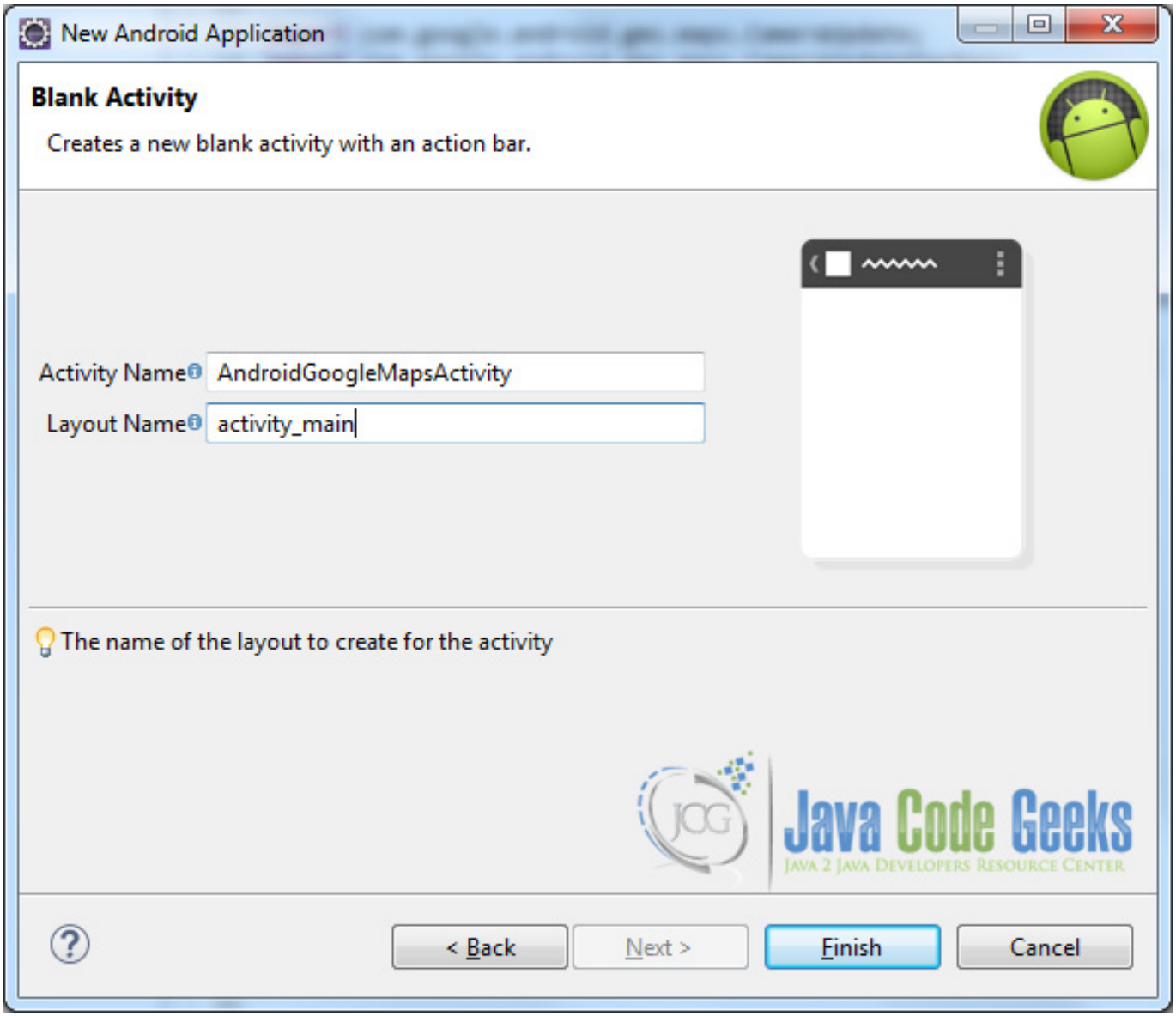

Figure 3.14: Create a new blank activity

Here you can see, how will the structure of the project become when finished:

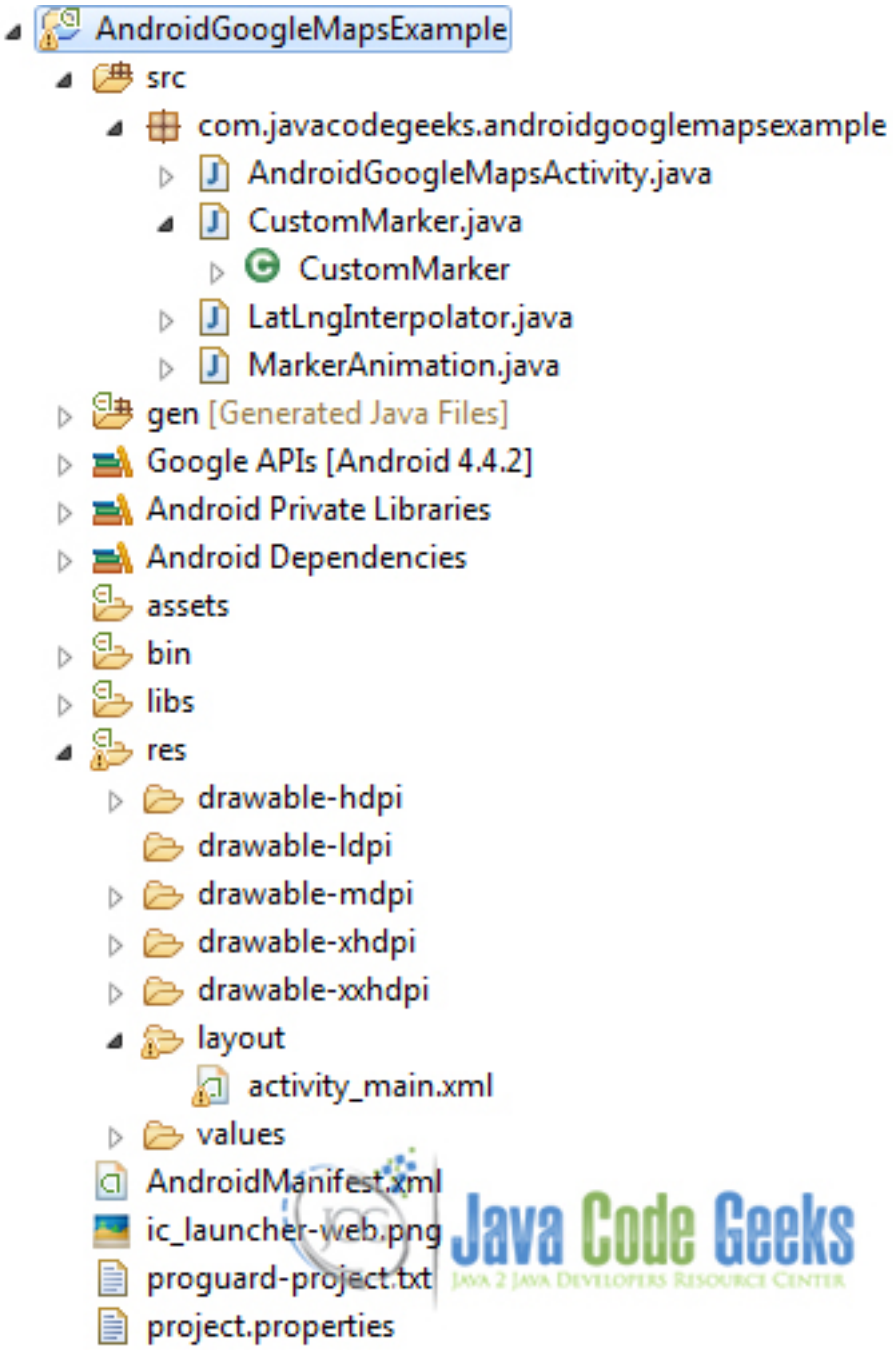

Figure 3.15: The tree of the project

3.3 Importing Google Play Services in your project

In order to be able to work with Google Maps v2, we have to import the Google Play Services library in the project.

Google Maps v2 will work only on devices that do have Google Play Services. If your mobile device does not have Google Play Services you have to download and use the Google Play Services application from Google Play.

Download Google Play Services SDK. We have to launch Android SDK Manager and install the Google Play services package.

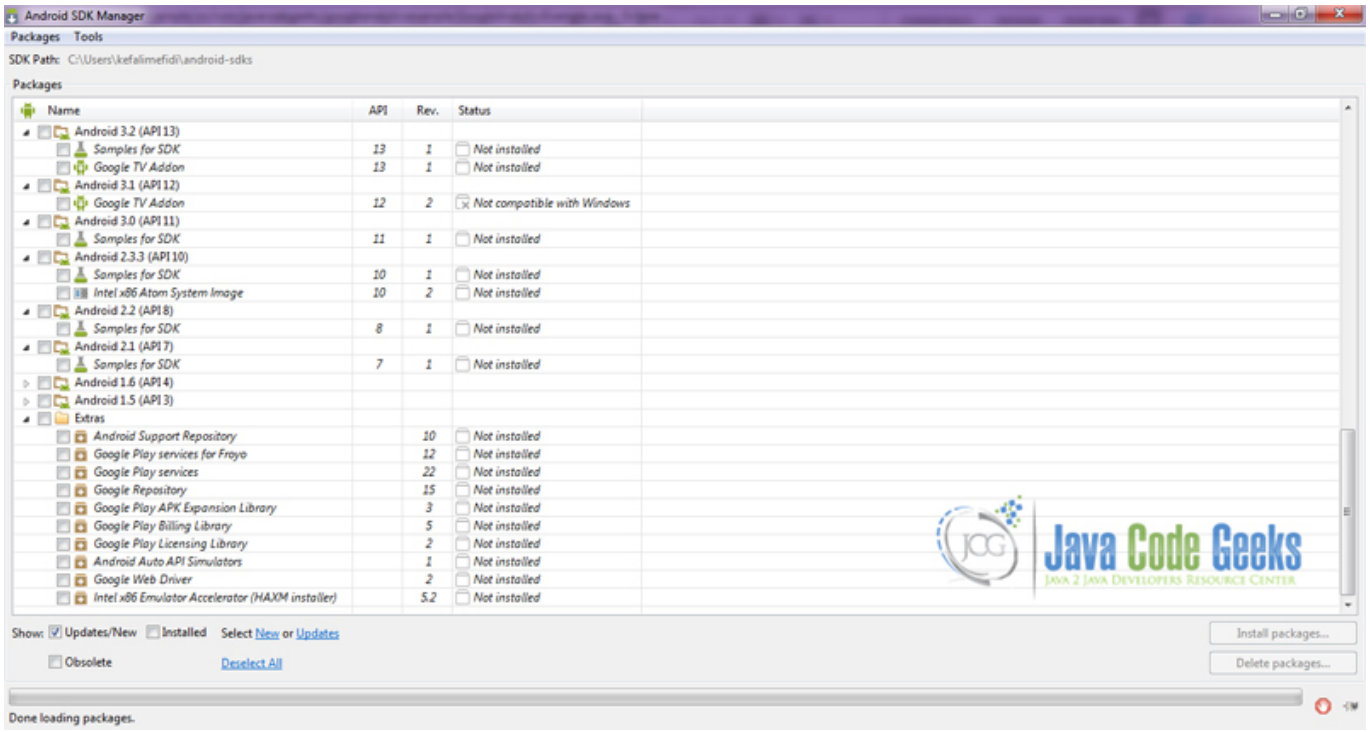

Figure 3.16: Launch Android SDK Manager

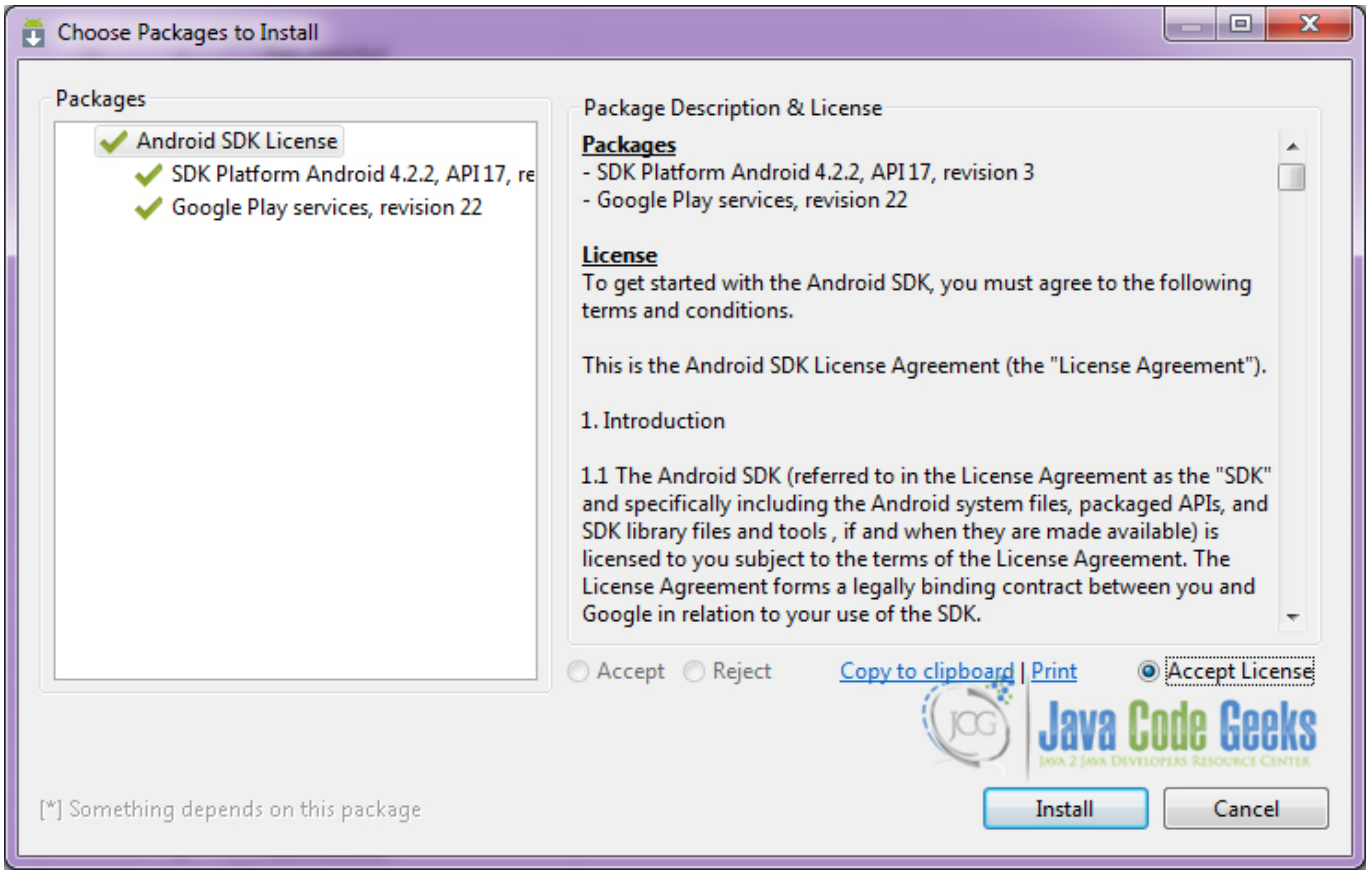

Figure 3.17: Download Google Play Services library

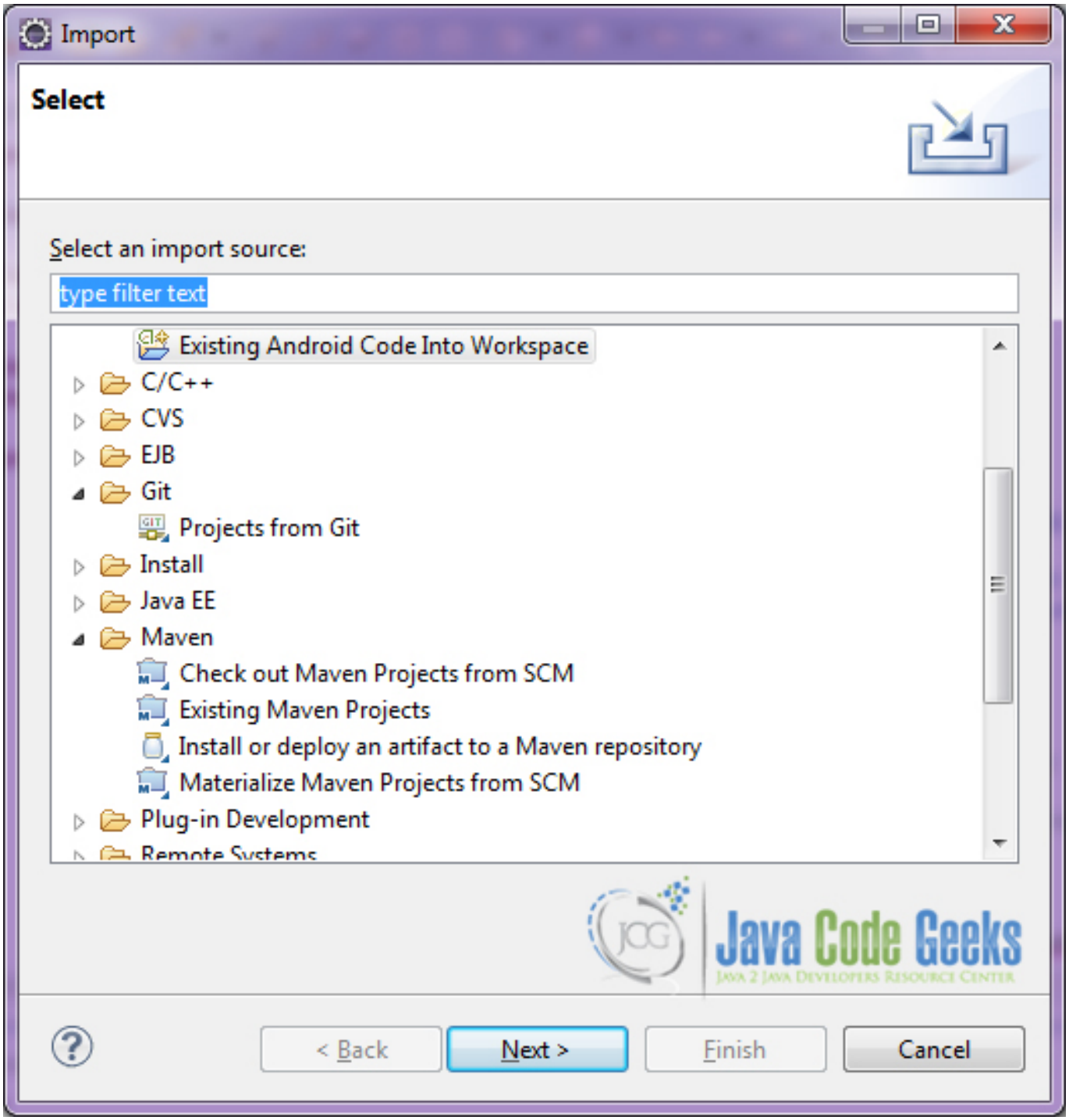

Figure 3.18: Import the library as an existing Android project

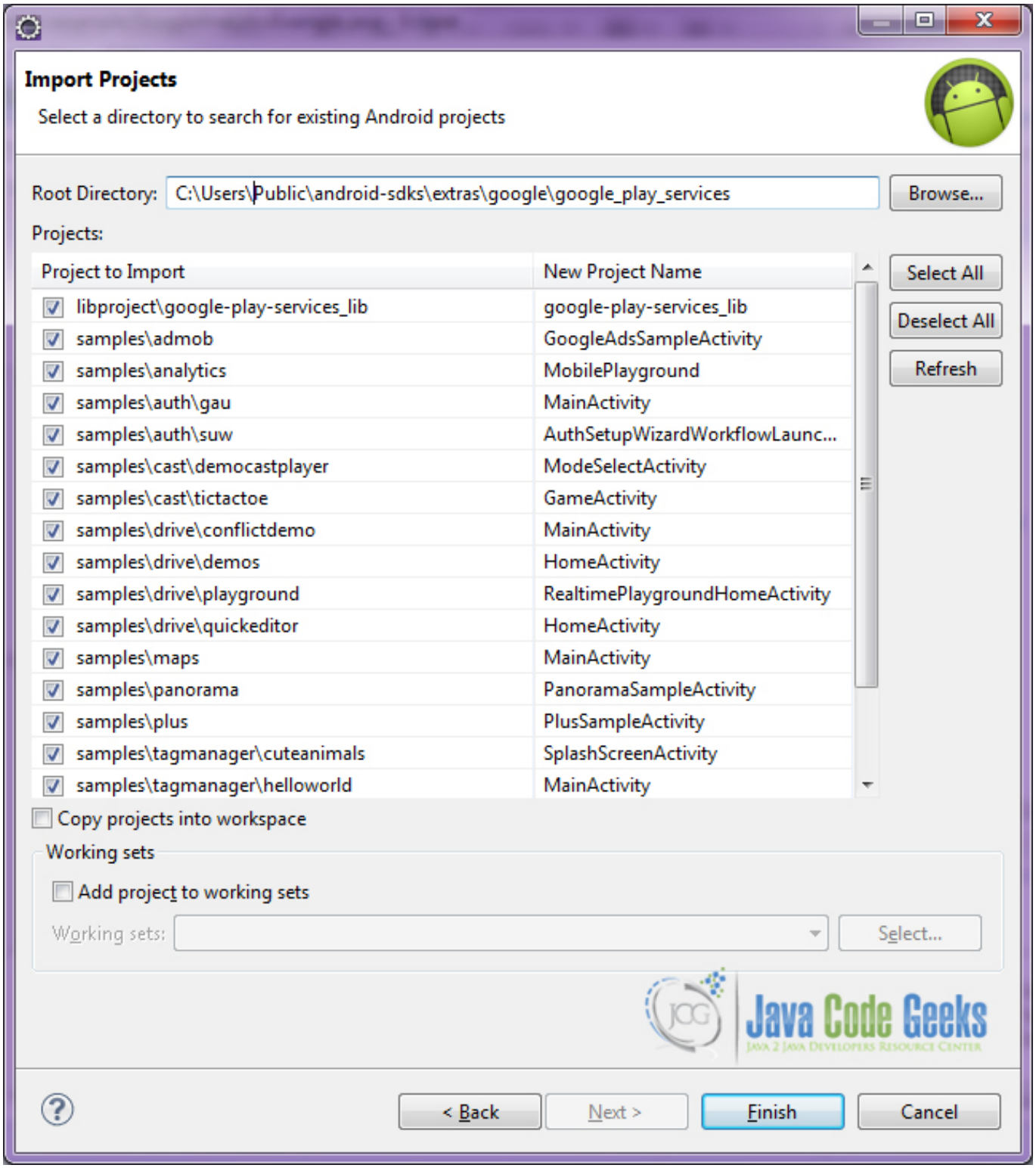

Figure 3.19: Import the library as an existing Android project

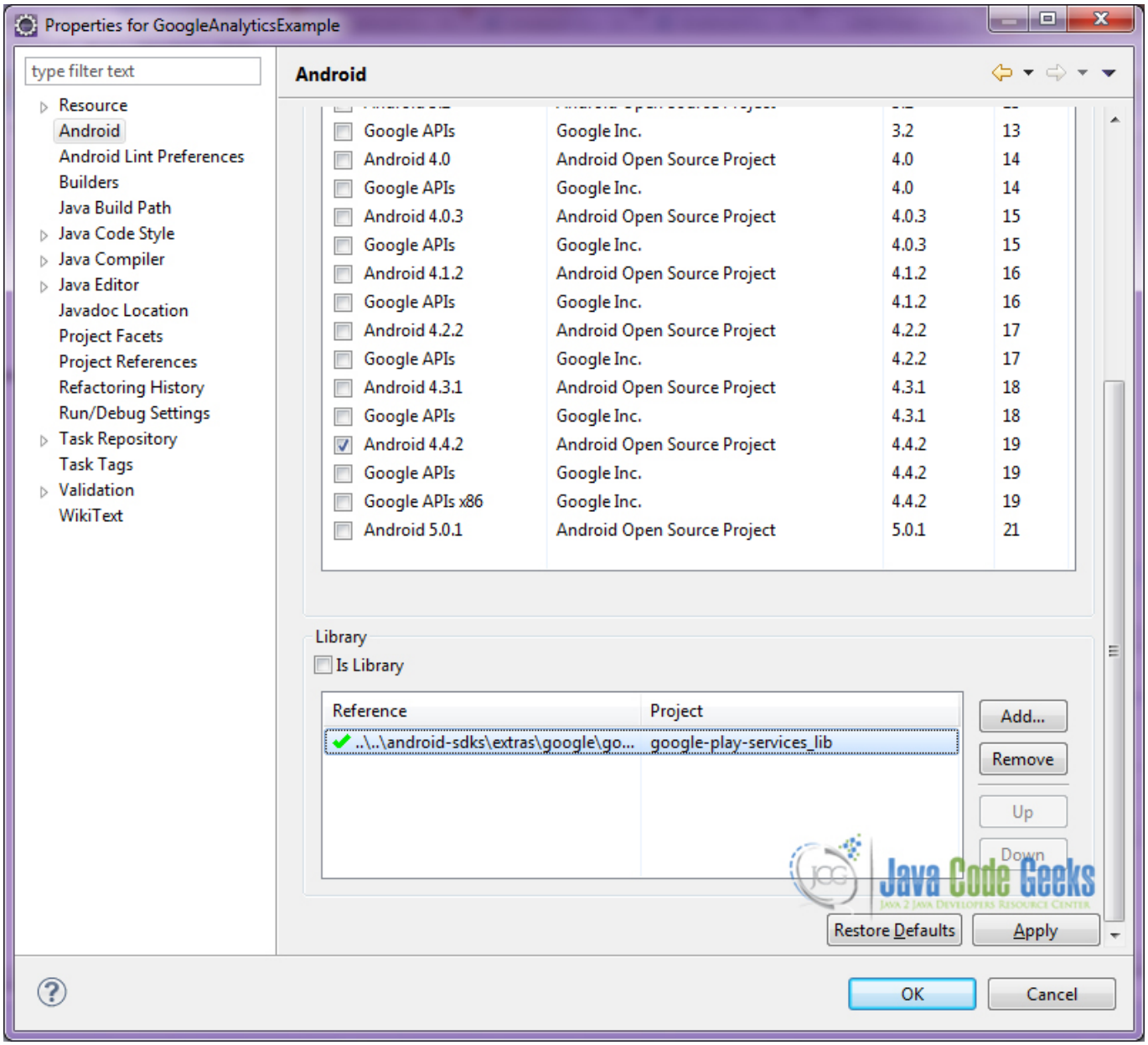

Figure 3.20: Import completed.

After downloading the Google Play Service package, just import this project into the main Android Google Maps v2 project.

3.4 Create the layout of the main Google Maps v2

We are going to make a simple layout xml for the `AndroidGoogleMapsExample.java`, that consists of a parent `RelativeLayout` and a child `LinearLayout` with vertical orientation, that includes a `Google Map Fragment` and a horizontal `LinearLayout` that includes 3 `Buttons` that lister to the corresponding actions.

Open `res/layout/activity_main.xml`, go to the respective xml tab and paste the following:

activity_main.xml

```
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
    xmlns:tools="https://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.javacodegeeks.androidgooglemapsexample.AndroidGoogleMapsActivity" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <fragment
            android:id="@+id/mapFragment"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            class="com.google.android.gms.maps.SupportMapFragment" />

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal" >

            <Button
                android:id="@+id/button1"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_weight="1"
                android:onClick="startAnimation"
                android:text="Animate Marker" />

            <Button
                android:id="@+id/button2"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_weight="1"
                android:onClick="zoomToMarkers"
                android:text="Zoom to markers" />

            <Button
                android:id="@+id/button3"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_weight="1"
                android:onClick="animateBack"
                android:text="Animate back to position" />
        </LinearLayout>
    </LinearLayout>

</RelativeLayout>
```

3.5 Create the source code of the main AndroidGoogleMapsActivity

Open `src/com.javacodegeeks.androidgooglemapsexample/AndroidGoogleMapsActivity.java` file and paste the code below.

AndroidGoogleMapsActivity.java

```
package com.javacodegeeks.androidgooglemapsexample;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.widget.Toast;

import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.LatLngBounds;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.javacodegeeks.androidgooglemapsexample.LatLngInterpolator.LinearFixed;

public class AndroidGoogleMapsActivity extends FragmentActivity {

        // Google Map
        private GoogleMap googleMap;
        private HashMap markersHashMap;
        private Iterator<Entry> iter;
        private CameraUpdate cu;
        private CustomMarker customMarkerOne, customMarkerTwo;

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

                try {
                        // Loading map
                        initilizeMap();
                        initializeUiSettings();
                        initializeMapLocationSettings();
                        initializeMapTraffic();
                        initializeMapType();
                        initializeMapViewSettings();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        @Override
        protected void onResume() {
                super.onResume();
                // initilizeMap();
        }

        private void initilizeMap() {

                googleMap = ((SupportMapFragment) getSupportFragmentManager(). ↩
                    findFragmentById(R.id.mapFragment)).getMap();

                // check if map is created successfully or not
                if (googleMap == null) {
```

```
                    Toast.makeText(getApplicationContext(), "Sorry! unable to create ↩
                        maps", Toast.LENGTH_SHORT).show();
            }

            (findViewById(R.id.mapFragment)).getViewTreeObserver(). ↩
                addOnGlobalLayoutListener(
                        new android.view.ViewTreeObserver.OnGlobalLayoutListener() ↩
                            {

                                @Override
                                public void onGlobalLayout() {
                                        if (android.os.Build.VERSION.SDK_INT >= 16) ↩
                                            {
                                                (findViewById(R.id.mapFragment)). ↩
                                                    getViewTreeObserver(). ↩
                                                    removeOnGlobalLayoutListener( ↩
                                                    this);
                                        } else {
                                                (findViewById(R.id.mapFragment)). ↩
                                                    getViewTreeObserver(). ↩
                                                    removeGlobalOnLayoutListener( ↩
                                                    this);
                                        }
                                        setCustomMarkerOnePosition();
                                        setCustomMarkerTwoPosition();
                                }
                        });
    }

    void setCustomMarkerOnePosition() {
            customMarkerOne = new CustomMarker("markerOne", 40.7102747, -73.9945297);
            addMarker(customMarkerOne);
    }

    void setCustomMarkerTwoPosition() {
            customMarkerTwo = new CustomMarker("markerTwo", 43.7297251, -74.0675716);
            addMarker(customMarkerTwo);
    }

    public void startAnimation(View v) {
            animateMarker(customMarkerOne, new LatLng(40.0675716, 40.7297251));
    }

    public void zoomToMarkers(View v) {
            zoomAnimateLevelToFitMarkers(120);
    }

    public void animateBack(View v) {
            animateMarker(customMarkerOne, new LatLng(32.0675716, 27.7297251));
    }

    public void initializeUiSettings() {
            googleMap.getUiSettings().setCompassEnabled(true);
            googleMap.getUiSettings().setRotateGesturesEnabled(false);
            googleMap.getUiSettings().setTiltGesturesEnabled(true);
            googleMap.getUiSettings().setZoomControlsEnabled(true);
            googleMap.getUiSettings().setMyLocationButtonEnabled(true);
    }

    public void initializeMapLocationSettings() {
            googleMap.setMyLocationEnabled(true);
    }
```

```
public void initializeMapTraffic() {
        googleMap.setTrafficEnabled(true);
}

public void initializeMapType() {
        googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
}


public void initializeMapViewSettings() {
        googleMap.setIndoorEnabled(true);
        googleMap.setBuildingsEnabled(false);
}


//this is method to help us set up a Marker that stores the Markers we want to plot ↩
    on the map
public void setUpMarkersHashMap() {
        if (markersHashMap == null) {
                markersHashMap = new HashMap();
        }
}

//this is method to help us add a Marker into the hashmap that stores the Markers
public void addMarkerToHashMap(CustomMarker customMarker, Marker marker) {
        setUpMarkersHashMap();
        markersHashMap.put(customMarker, marker);
}

//this is method to help us find a Marker that is stored into the hashmap
public Marker findMarker(CustomMarker customMarker) {
        iter = markersHashMap.entrySet().iterator();
        while (iter.hasNext()) {
                Map.Entry mEntry = (Map.Entry) iter.next();
                CustomMarker key = (CustomMarker) mEntry.getKey();
                if (customMarker.getCustomMarkerId().equals(key.getCustomMarkerId() ↩
                    )) {
                        Marker value = (Marker) mEntry.getValue();
                        return value;
                }
        }
        return null;
}


//this is method to help us add a Marker to the map
public void addMarker(CustomMarker customMarker) {
        MarkerOptions markerOption = new MarkerOptions().position(
                        new LatLng(customMarker.getCustomMarkerLatitude(),  ↩
                            customMarker.getCustomMarkerLongitude())).icon(
                        BitmapDescriptorFactory.defaultMarker());

        Marker newMark = googleMap.addMarker(markerOption);
        addMarkerToHashMap(customMarker, newMark);
}

//this is method to help us remove a Marker
public void removeMarker(CustomMarker customMarker) {
        if (markersHashMap != null) {
                if (findMarker(customMarker) != null) {
                        findMarker(customMarker).remove();
```

```
                        markersHashMap.remove(customMarker);
                }
        }
}

//this is method to help us fit the Markers into specific bounds for camera position
public void zoomAnimateLevelToFitMarkers(int padding) {
        LatLngBounds.Builder b = new LatLngBounds.Builder();
        iter = markersHashMap.entrySet().iterator();

        while (iter.hasNext()) {
                Map.Entry mEntry = (Map.Entry) iter.next();
                CustomMarker key = (CustomMarker) mEntry.getKey();
                LatLng ll = new LatLng(key.getCustomMarkerLatitude(), key.getCustomMarkerLongitude());
                b.include(ll);
        }
        LatLngBounds bounds = b.build();

        // Change the padding as per needed
        cu = CameraUpdateFactory.newLatLngBounds(bounds, padding);
        googleMap.animateCamera(cu);
}

//this is method to help us move a Marker.
public void moveMarker(CustomMarker customMarker, LatLng latlng) {
        if (findMarker(customMarker) != null) {
                findMarker(customMarker).setPosition(latlng);
                customMarker.setCustomMarkerLatitude(latlng.latitude);
                customMarker.setCustomMarkerLongitude(latlng.longitude);
        }
}

//this is method to animate the Marker. There are flavours for all Android versions
public void animateMarker(CustomMarker customMarker, LatLng latlng) {
        if (findMarker(customMarker) != null) {

                LatLngInterpolator latlonInter = new LinearFixed();
                latlonInter.interpolate(20,
                                new LatLng(customMarker.getCustomMarkerLatitude(), customMarker.getCustomMarkerLongitude()), latlng
                                );

                customMarker.setCustomMarkerLatitude(latlng.latitude);
                customMarker.setCustomMarkerLongitude(latlng.longitude);

                if (android.os.Build.VERSION.SDK_INT >= 14) {
                        MarkerAnimation.animateMarkerToICS(findMarker(customMarker), new LatLng(customMarker.getCustomMarkerLatitude(),
                                        customMarker.getCustomMarkerLongitude()), latlonInter);
                } else if (android.os.Build.VERSION.SDK_INT >= 11) {
                        MarkerAnimation.animateMarkerToHC(findMarker(customMarker), new LatLng(customMarker.getCustomMarkerLatitude(),
                                        customMarker.getCustomMarkerLongitude()), latlonInter);
                } else {
                        MarkerAnimation.animateMarkerToGB(findMarker(customMarker), new LatLng(customMarker.getCustomMarkerLatitude(),
                                        customMarker.getCustomMarkerLongitude()), latlonInter);
```

```
                        }
                }
        }

}
```

Let's take a closer look:

```
private void initilizeMap() {

                googleMap = ((SupportMapFragment) getSupportFragmentManager(). ↩
                    findFragmentById(R.id.mapFragment)).getMap();

                if (googleMap == null) {
                        Toast.makeText(getApplicationContext(), "Sorry! unable to create  ↩
                            maps", Toast.LENGTH_SHORT).show();
                }

                (findViewById(R.id.mapFragment)).getViewTreeObserver(). ↩
                    addOnGlobalLayoutListener(
                                new android.view.ViewTreeObserver.OnGlobalLayoutListener()  ↩
                                    {

                                        @Override
                                        public void onGlobalLayout() {
                                                // gets called after layout has been done  ↩
                                                    but before display
                                                // so we can get the height then hide the  ↩
                                                    view
                                                if (android.os.Build.VERSION.SDK_INT >= 16) ↩
                                                     {
                                                        (findViewById(R.id.mapFragment)). ↩
                                                            getViewTreeObserver(). ↩
                                                            removeOnGlobalLayoutListener( ↩
                                                            this);
                                                } else {
                                                        (findViewById(R.id.mapFragment)). ↩
                                                            getViewTreeObserver(). ↩
                                                            removeGlobalOnLayoutListener( ↩
                                                            this);
                                                }
                                                setCustomMarkerOnePosition();
                                                setCustomMarkerTwoPosition();
                                                // plotMarkers(markerList);
                                        }
                                });
        }
```

With this method `initilizeMap()`, we set up our map in the `Fragment`. We use `SupportMapFragment` in order to make our map compatible with versions older than the Android version 4.0. Also by using the `OnGlobalLayoutListener` that is called after layout has been done, but before display, in order to avoid making any changes to the map, before the map is fully set up in our layout. Otherwise an error that says, that map is not ready yet, may appear.

```
public void initializeUiSettings() {
                googleMap.getUiSettings().setCompassEnabled(true);
                googleMap.getUiSettings().setRotateGesturesEnabled(false);
                googleMap.getUiSettings().setTiltGesturesEnabled(true);
                googleMap.getUiSettings().setZoomControlsEnabled(true);
                googleMap.getUiSettings().setMyLocationButtonEnabled(true);
        }
```

With the method above, we initialize the map Ui settings, we set the compass, the gestures, the tilt gestures, the zoom controls and the button that centers us to our location spot.

```
public void initializeMapLocationSettings() {
                googleMap.setMyLocationEnabled(true);
        }
```

With the method above, we enable the blue location spot, that shows our position on the map.

```
public void initializeMapTraffic() {
                googleMap.setTrafficEnabled(true);
        }
```

With the method above, we enable the traffic lines layer on the map.

```
public void initializeMapType() {
                googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
        }
```

With the method above, we set the Map Type to the `GoogleMap.MAP_TYPE_NORMAL`.

```
        public void initializeMapViewSettings() {
                googleMap.setIndoorEnabled(true);
                googleMap.setBuildingsEnabled(false);
        }
```

With the method above, we initialize the Map View settings, meaning the basic view map and buildings settings.

```
public void zoomAnimateLevelToFitMarkers(int padding) {
                LatLngBounds.Builder b = new LatLngBounds.Builder();
                iter = markersHashMap.entrySet().iterator();

                while (iter.hasNext()) {
                        Map.Entry mEntry = (Map.Entry) iter.next();
                        CustomMarker key = (CustomMarker) mEntry.getKey();
                        LatLng ll = new LatLng(key.getCustomMarkerLatitude(), key. ↩
                            getCustomMarkerLongitude());
                        b.include(ll);
                }
                LatLngBounds bounds = b.build();

                // Change the padding as per needed
                cu = CameraUpdateFactory.newLatLngBounds(bounds, padding);
                googleMap.animateCamera(cu);
        }
```

With the above method, we want to zoom the map, at a zoom that we will be able to view every marker that we added on the map. So, this is method to help us fit an array of Markers into specific bounds for camera position.

3.6 Creating the source code of the helper class CustomMarker.java

We are going to create the `CustomMarker.java` in order to help us make multiple markers and store them in an array, so that we can use them in the `AndroidGoogleMapsActivity`. This is a helper class.

Open `src/com.javacodegeeks.androidgooglemapsexample/CustomMarker.java` file and paste the code below.

CustomMarker.java

```
package com.javacodegeeks.androidgooglemapsexample;

public class CustomMarker {

        private String id;
        private Double latitude;
        private Double longitude;

        public CustomMarker(String id, Double latitude, Double longitude) {

                this.id = id;
                this.latitude = latitude;
                this.longitude = longitude;
        }

        public CustomMarker() {
                this.id = "";
                this.latitude = 0.0;
                this.longitude = 0.0;
        }

        public String getCustomMarkerId() {
                return id;
        }

        public void setCustomMarkerId(String id) {
                this.id = id;
        }

        public Double getCustomMarkerLatitude() {
                return latitude;
        }

        public void setCustomMarkerLatitude(Double mLatitude) {
                this.latitude = mLatitude;
        }

        public Double getCustomMarkerLongitude() {
                return longitude;
        }

        public void setCustomMarkerLongitude(Double mLongitude) {
                this.longitude = mLongitude;
        }
}
```

As you can see below, we have added an id attribute in our helper class, in order to have an identifier for our CustomMarker. This will help us find the right marker that we have created and change, or remove it separately.

```
public void setCustomMarkerId(String id) {
                this.id = id;
        }
```

3.7 Creating the source code of the helper class LatLngInterpolator.java

Open `src/com.javacodegeeks.androidgooglemapsexample/LatLngInterpolator.java` file and paste the code below.

LatLngInterpolator.java

```
package com.javacodegeeks.androidgooglemapsexample;

import static java.lang.Math.asin;
import static java.lang.Math.atan2;
import static java.lang.Math.cos;
import static java.lang.Math.pow;
import static java.lang.Math.sin;
import static java.lang.Math.sqrt;
import static java.lang.Math.toDegrees;
import static java.lang.Math.toRadians;

import com.google.android.gms.maps.model.LatLng;

public interface LatLngInterpolator {
        public LatLng interpolate(float fraction, LatLng a, LatLng b);

        public class Linear implements LatLngInterpolator {
                @Override
                public LatLng interpolate(float fraction, LatLng a, LatLng b) {
                        double lat = (b.latitude - a.latitude) * fraction + a.latitude;
                        double lng = (b.longitude - a.longitude) * fraction + a.longitude;
                        return new LatLng(lat, lng);
                }
        }

        public class LinearFixed implements LatLngInterpolator {
                @Override
                public LatLng interpolate(float fraction, LatLng a, LatLng b) {
                        double lat = (b.latitude - a.latitude) * fraction + a.latitude;
                        double lngDelta = b.longitude - a.longitude;

                        if (Math.abs(lngDelta) > 180) {
                                lngDelta -= Math.signum(lngDelta) * 360;
                        }
                        double lng = lngDelta * fraction + a.longitude;
                        return new LatLng(lat, lng);
                }
        }

        public class Spherical implements LatLngInterpolator {

                @Override
                public LatLng interpolate(float fraction, LatLng from, LatLng to) {

                        double fromLat = toRadians(from.latitude);
                        double fromLng = toRadians(from.longitude);
                        double toLat = toRadians(to.latitude);
                        double toLng = toRadians(to.longitude);
                        double cosFromLat = cos(fromLat);
                        double cosToLat = cos(toLat);

                        double angle = computeAngleBetween(fromLat, fromLng, toLat, toLng);
                        double sinAngle = sin(angle);
                        if (sinAngle < 1E-6) {
                                return from;
                        }
                        double a = sin((1 - fraction) * angle) / sinAngle;
                        double b = sin(fraction * angle) / sinAngle;

                        double x = a * cosFromLat * cos(fromLng) + b * cosToLat * cos(toLng ↩
                            );
```

```
                        double y = a * cosFromLat * sin(fromLng) + b * cosToLat * sin(toLng ↩
                            );
                        double z = a * sin(fromLat) + b * sin(toLat);

                        double lat = atan2(z, sqrt(x * x + y * y));
                        double lng = atan2(y, x);
                        return new LatLng(toDegrees(lat), toDegrees(lng));
                }

                private double computeAngleBetween(double fromLat, double fromLng, double  ↩
                    toLat, double toLng) {

                        double dLat = fromLat - toLat;
                        double dLng = fromLng - toLng;
                        return 2 * asin(sqrt(pow(sin(dLat / 2), 2) + cos(fromLat) * cos( ↩
                            toLat) * pow(sin(dLng / 2), 2)));
                }
        }
}
```

3.8 Creating the source code of the helper class MarkerAnimation.java

Open `src/com.javacodegeeks.androidgooglemapsexample/MarkerAnimation.java` file and paste the code below.

MarkerAnimation.java

```
package com.javacodegeeks.androidgooglemapsexample;

import android.animation.ObjectAnimator;
import android.animation.TypeEvaluator;
import android.animation.ValueAnimator;
import android.annotation.TargetApi;
import android.os.Build;
import android.os.Handler;
import android.os.SystemClock;
import android.util.Property;
import android.view.animation.AccelerateDecelerateInterpolator;
import android.view.animation.Interpolator;

import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;

public class MarkerAnimation {

        static void animateMarkerToGB(final Marker marker, final LatLng finalPosition,  ↩
            final LatLngInterpolator latLngInterpolator) {
                final LatLng startPosition = marker.getPosition();
                final Handler handler = new Handler();
                final long start = SystemClock.uptimeMillis();
                final Interpolator interpolator = new AccelerateDecelerateInterpolator();
                final float durationInMs = 500;

                handler.post(new Runnable() {
                        long elapsed;
                        float t;
                        float v;

                        @Override
                        public void run() {
```

```
                                // Calculate progress using interpolator
                                elapsed = SystemClock.uptimeMillis() - start;
                                t = elapsed / durationInMs;
                                v = interpolator.getInterpolation(t);

                                marker.setPosition(latLngInterpolator.interpolate(v,
                                    startPosition, finalPosition));

                                if (t < 1) {

                                        handler.postDelayed(this, 16);
                                }
                        }
                });
        }

        @TargetApi(Build.VERSION_CODES.HONEYCOMB)
        static void animateMarkerToHC(final Marker marker, final LatLng finalPosition,
            final LatLngInterpolator latLngInterpolator) {
                final LatLng startPosition = marker.getPosition();

                ValueAnimator valueAnimator = new ValueAnimator();
                valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener()
                    {
                        @Override
                        public void onAnimationUpdate(ValueAnimator animation) {
                                float v = animation.getAnimatedFraction();
                                LatLng newPosition = latLngInterpolator.interpolate(v,
                                    startPosition, finalPosition);
                                marker.setPosition(newPosition);
                        }
                });
                valueAnimator.setFloatValues(0, 1);
                valueAnimator.setDuration(500);
                valueAnimator.start();
        }

        @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH)
        static void animateMarkerToICS(Marker marker, LatLng finalPosition, final
            LatLngInterpolator latLngInterpolator) {
                TypeEvaluator typeEvaluator = new TypeEvaluator() {
                        @Override
                        public LatLng evaluate(float fraction, LatLng startValue, LatLng
                            endValue) {
                                return latLngInterpolator.interpolate(fraction, startValue,
                                     endValue);
                        }
                };
                Property property = Property.of(Marker.class, LatLng.class, "position");
                ObjectAnimator animator = ObjectAnimator.ofObject(marker, property,
                    typeEvaluator, finalPosition);
                animator.setDuration(500);
                animator.start();
        }
}
```

3.9 Modifying the AndroidManifest.xml

In order to make our map visible, we should have access to the Internet. This must be specified in the manifest, so that, our application will be granted the permission to use the Internet connection, with **INTERNET**, **ACCESS_NETWORK_STATE** and **WRITE_EXTERNAL_STORAGE**. Also, a good idea is to give permission **ACCESS_COARSE_LOCATION** and **ACCESS_FINE_LOCATION** for better Location results.

Also, in order to add and use the Google Play Services library, we have to add the specific `meta-data` tag in our xml. Also, we should not forget the **API KEY** `meta-data` tag in our xml, in order to get permission to use Google Maps.

The AndroidManifest.xml of our project is simple and contains the permissions:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.androidgooglemapsexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="19" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <!--
     The following two permissions are not required to use
     Google Maps Android API v2, but are recommended.
    -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".AndroidGoogleMapsActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyAU9ShujnIg3IDQxtPrafaf7Q1qOvFVdwNmWc4" />
    </application>

</manifest>
```

3.10 Build, compile and run

When we build, compile and run our project, the main AndroidGoogleMapsExample should look like this:

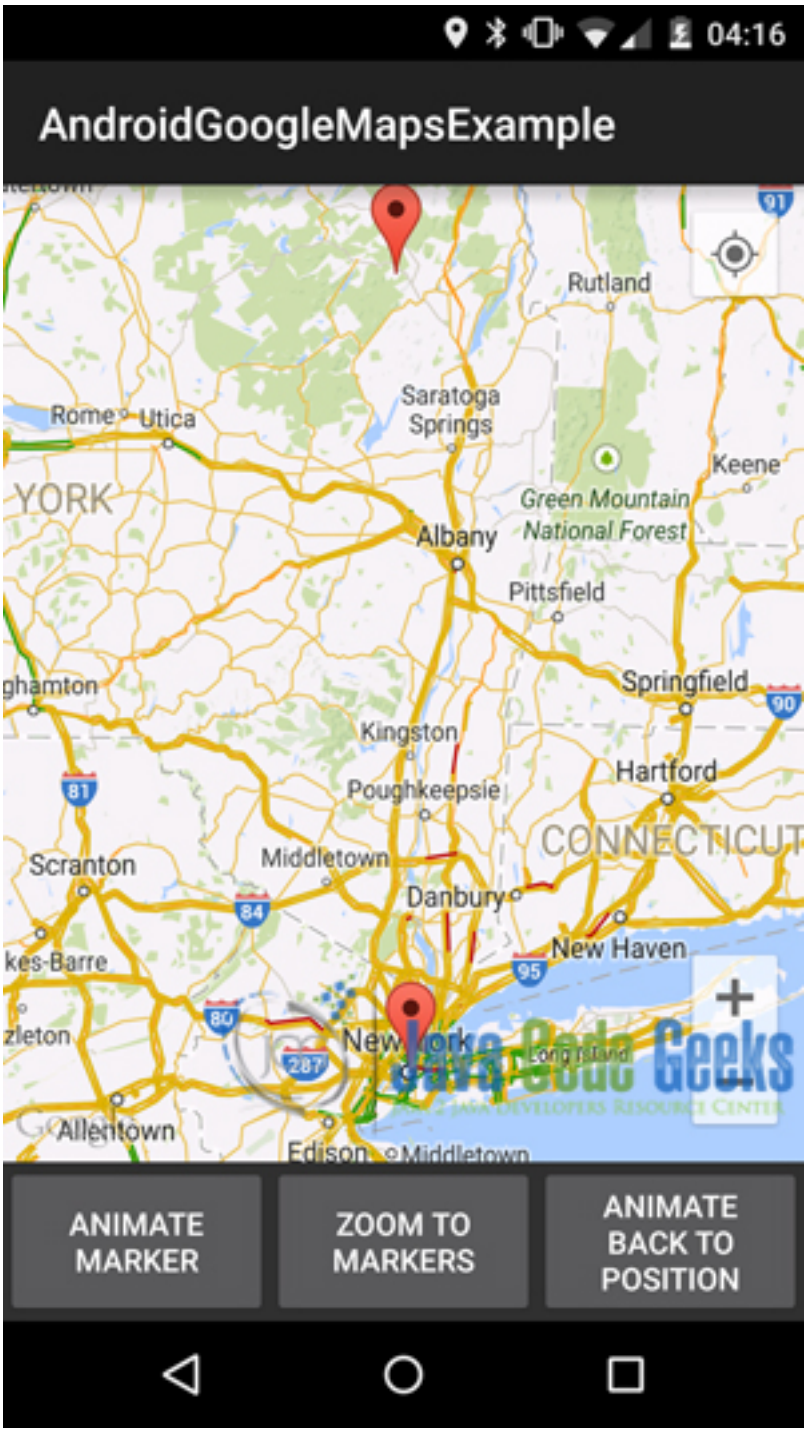

Figure 3.21: This is how the main Activity with the Google maps Fragment looks like

Figure 3.22: This is how the Google Map v2 looks like after the first animation

Figure 3.23: This is how the Google Map v2 looks like after the second animation

3.11 Download the Eclipse Project

This was the example of Android Google Maps v2.

Download You can download the full source code of this example here: **AndroidGoogleMapsExample**

Chapter 4

Android Start Service on Boot Example

In the mobile device world, many times, we may need to make an application that has to do some tasks in the background, without the users to have to open their application. Some easy examples are the background Location tracking, or an alarm application, or the applications that want to receive push events from a server.

This can be done with the use of a service that will run in the background and will start on device boot, or on application update in a device. In order to achieve this, we are going to register a BroadcastReceiver to listen to `RECEIVE_BOOT_COMPLETED` events and then start a service to do whatever we may want our service to do.

So, in this example, we are going to show how to make an application that has an Android service that starts service at device boot.

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Android Studio 1.3.2
- Android SDK 5.0

Let's take a closer look:

4.1 Create a New Android Studio Project

Open Android Studio and choose Start a new Android Studio Project in the welcome screen.

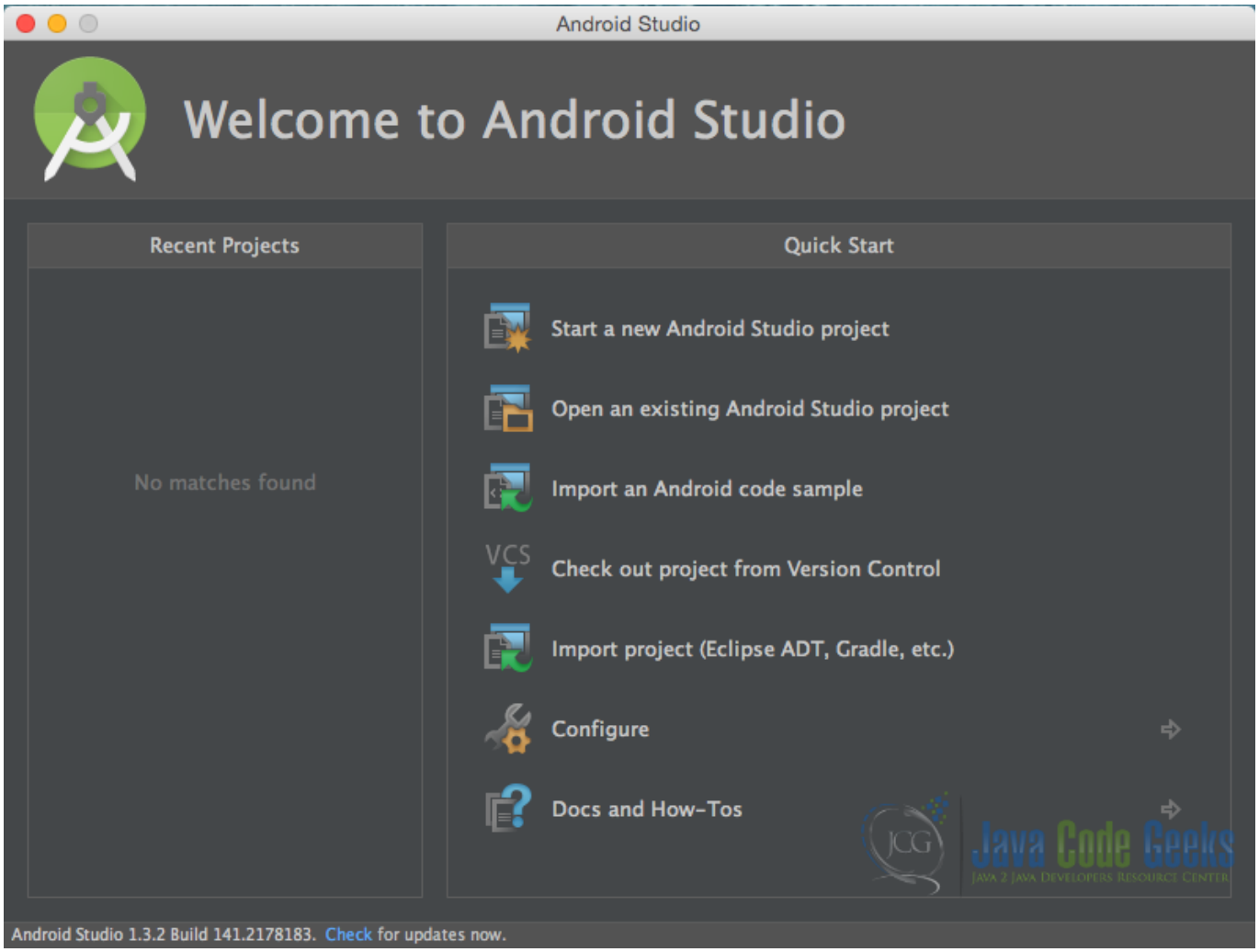

Figure 4.1: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

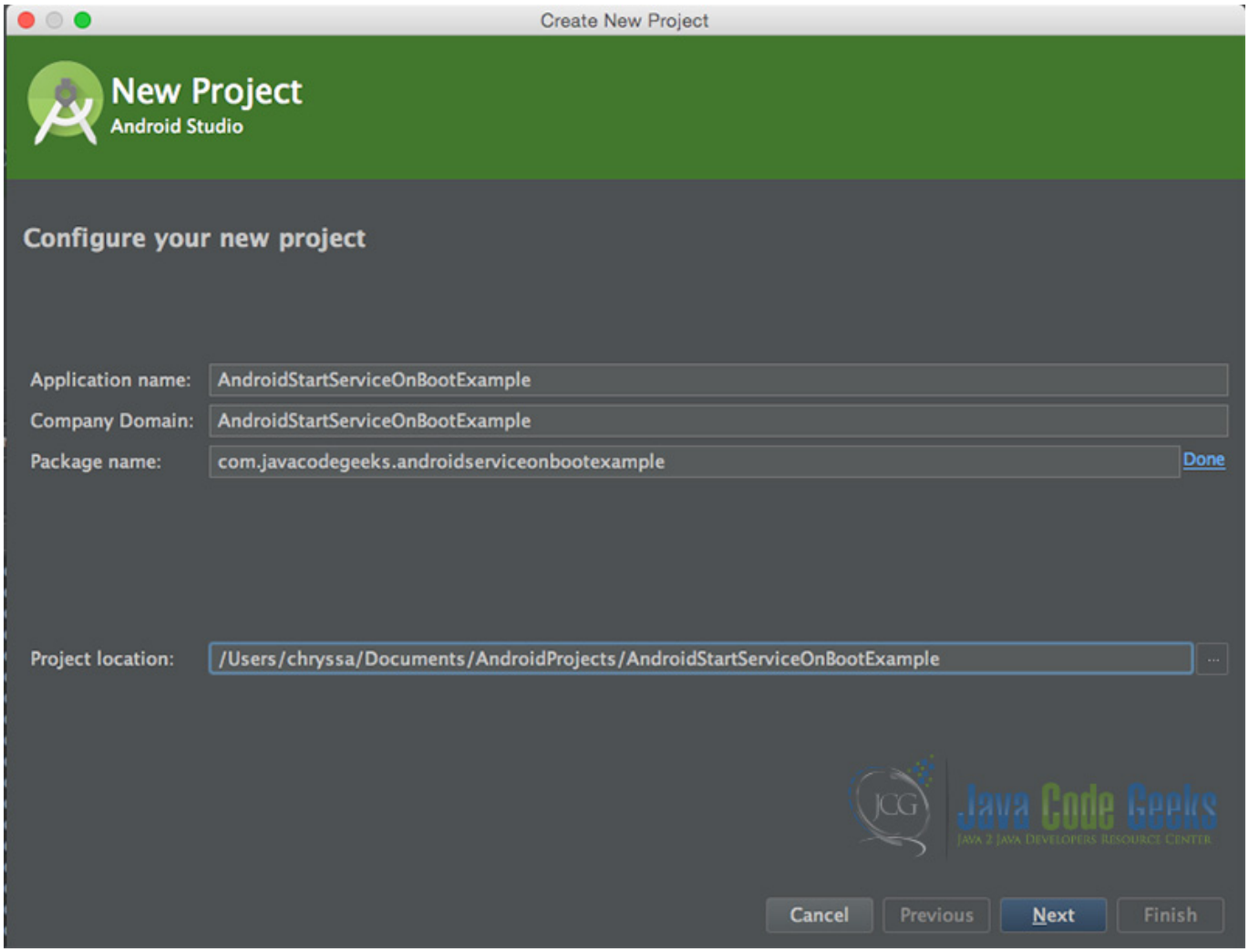

Figure 4.2: Configure your new project screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

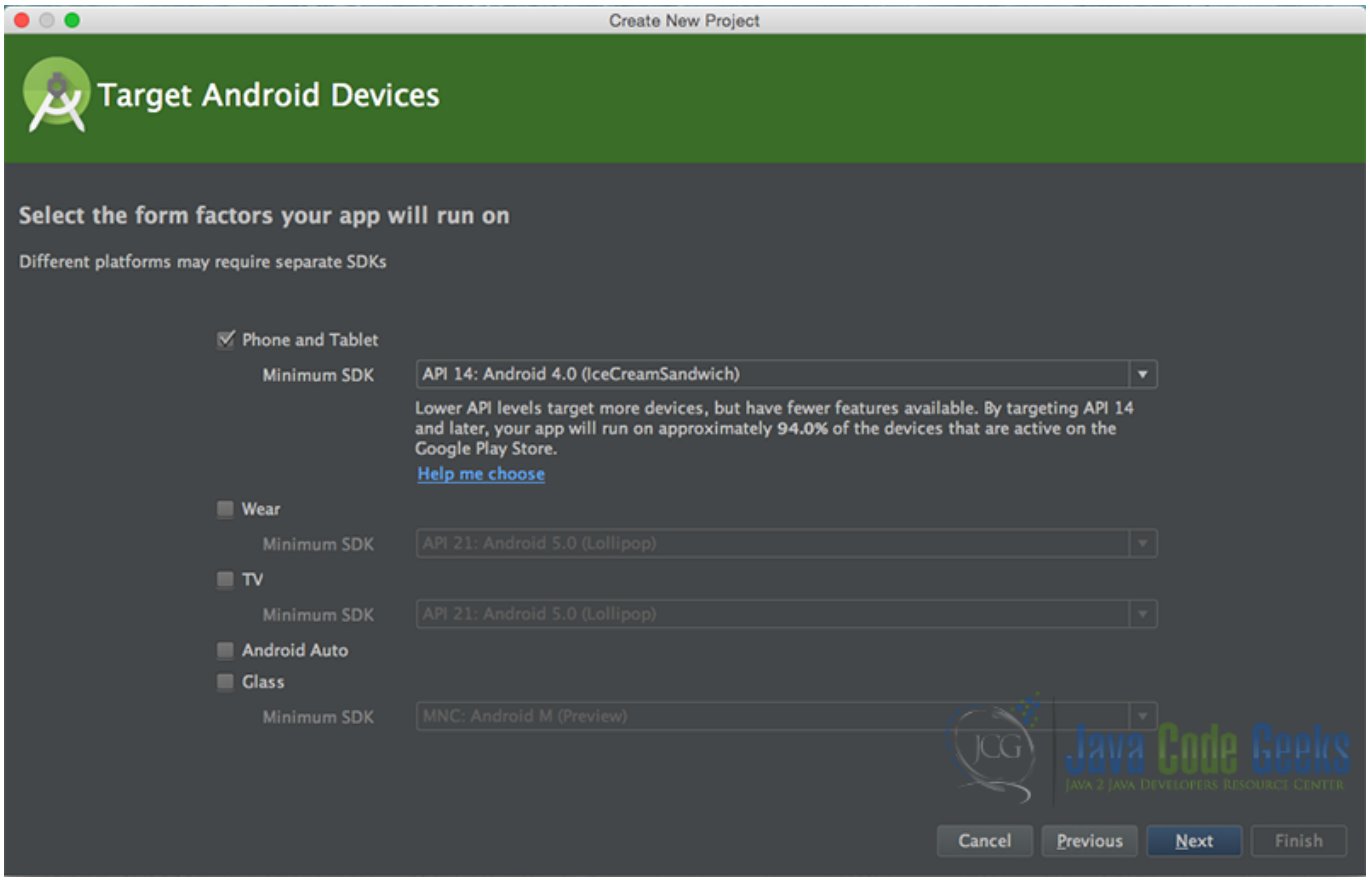

Figure 4.3: Target Android Devices screen.

In the next window you should choose Add no activity. In this example, we are going to create our Activity.

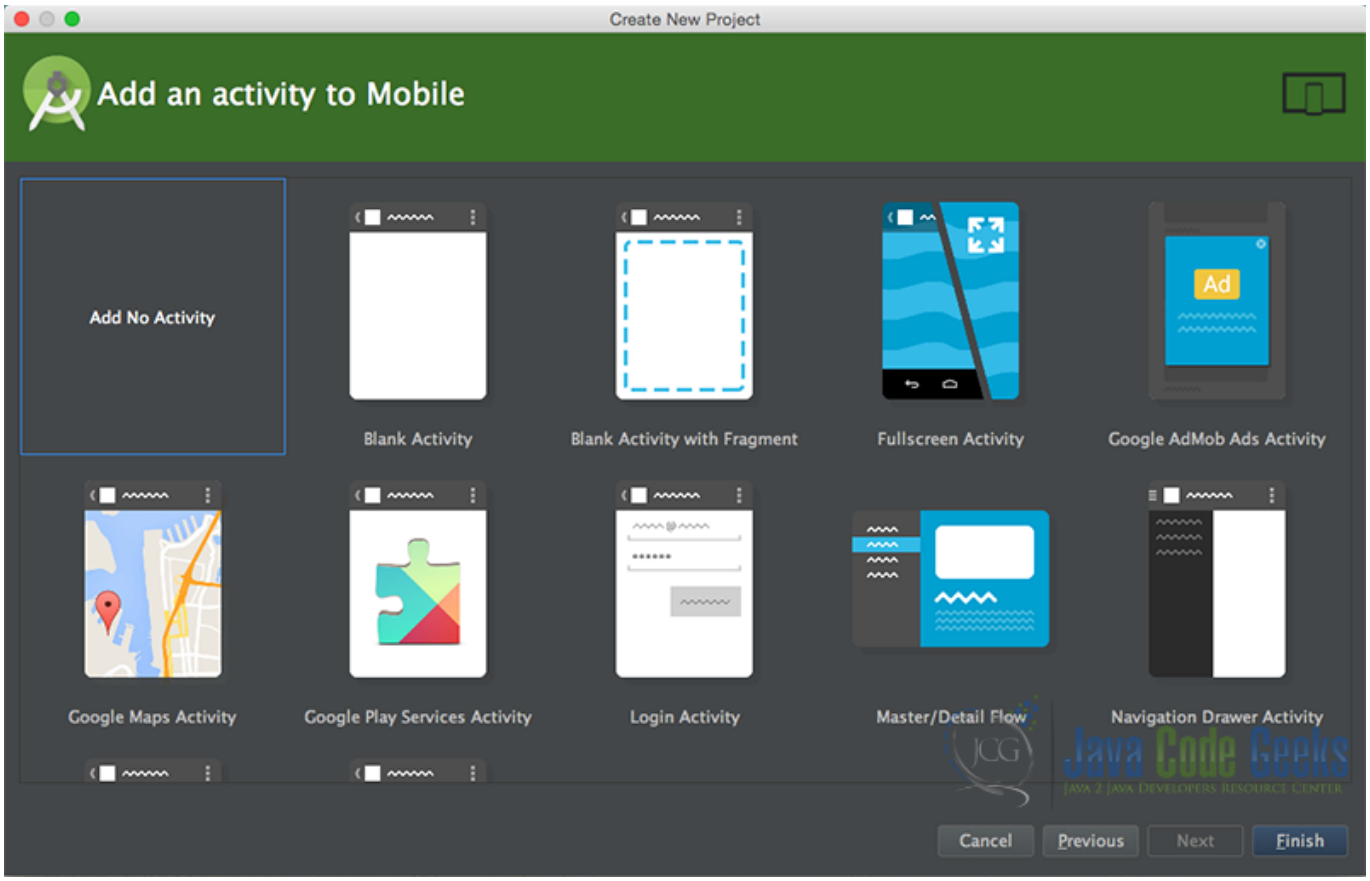

Figure 4.4: Add an activity to Mobile. Choose: Add no activity.

Now, our project has just been created!

4.2 Create the layout and the source code of a simple AndroidStartServiceOnBoot Activity

Add a new Java class Activity inside `src/com.javacodegeeks.androidserviceonbootexample/` so that we are going to have the `src/com.javacodegeeks.androidserviceonbootexample/AndroidCustomFontExample.java` file and paste the code below.

MainActivity.java

```
package com.javacodegeeks.androidserviceonbootexample;

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

public class MainActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

main.xml

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="This Activity does nothing at all. But an auto-start service has just ↩
            started!"
        android:textColor="#272727"
        android:textSize="20dp" />
</LinearLayout>
```

Actually, this activity will not do anything concerning the service that starts on boot. All the logic regarding the service is explained in the next lines.

4.3 Creating the source code of the BroadcastReceiverOnBootComplete Service

Add a new Java class inside `src/com.javacodegeeks.androidserviceonbootexample/` so that we are going to have the `src/com.javacodegeeks.androidserviceonbootexample/BroadcastReceiverOnBootComplete.java` file and paste the code below.

BroadcastReceiverOnBootComplete.java

```
package com.javacodegeeks.androidserviceonbootexample;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class BroadcastReceiverOnBootComplete extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equalsIgnoreCase(Intent.ACTION_BOOT_COMPLETED)) {
            Intent serviceIntent = new Intent(context, AndroidServiceStartOnBoot.class);
            context.startService(serviceIntent);
        }
    }
}
```

We have just created a BroadcastReceiver that will receive the `ACTION_BOOT_COMPLETED` intent. This means that when we boot up our device this class will "catch" the event and start the `AndroidServiceStartOnBoot` service.

4.4 Creating the source code of the AndroidServiceStartOnBoot Service

Add a new Java class inside `src/com.javacodegeeks.androidserviceonbootexample/` so that we are going to have the `src/com.javacodegeeks.androidserviceonbootexample/AndroidServiceStartOnBoot.java` file and paste the code below.

AndroidServiceStartOnBoot.java

```
package com.javacodegeeks.androidserviceonbootexample;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class AndroidServiceStartOnBoot extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
       // here you can add whatever you want this service to do
    }

}
```

4.5 Editing the Android Manifest xml

The `AndroidManifest.xml` of our project is :

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.androidserviceonbootexample">

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses- ↩
        permission>

    <receiver
        android:name="com.javacodegeeks.androidserviceonbootexample. ↩
            BroadcastReceiverOnBootComplete"
        android:enabled="true"
        android:exported="false">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.PACKAGE_REPLACED" />
            <data android:scheme="package" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.PACKAGE_ADDED" />
            <data android:scheme="package" />
        </intent-filter>
    </receiver>

    <service android:name="com.javacodegeeks.androidserviceonbootexample. ↩
        AndroidServiceStartOnBoot"></service>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
        android:theme="@style/AppTheme">

        <activity
            android:name="com.javacodegeeks.androidserviceonbootexample.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest
>
```

In this `AndroidManifest.xml` we register the application to listen to `android.intent.action.BOOT_COMPLETED`. Additionally, we register the `android.intent.action.PACKAGE_REPLACED` and the `android.intent.action.PACKAGE_ADDED` events. These events can catch the installation and update of our application, so that our service will start not only on device boot, but on application update.

We also have to register our service in the `AndroidManifest.xml`. This happens at line 24.

4.6 Build, compile and run

When we build, compile and run our project, the main Android ServiceStart OnBoot application should look like this:

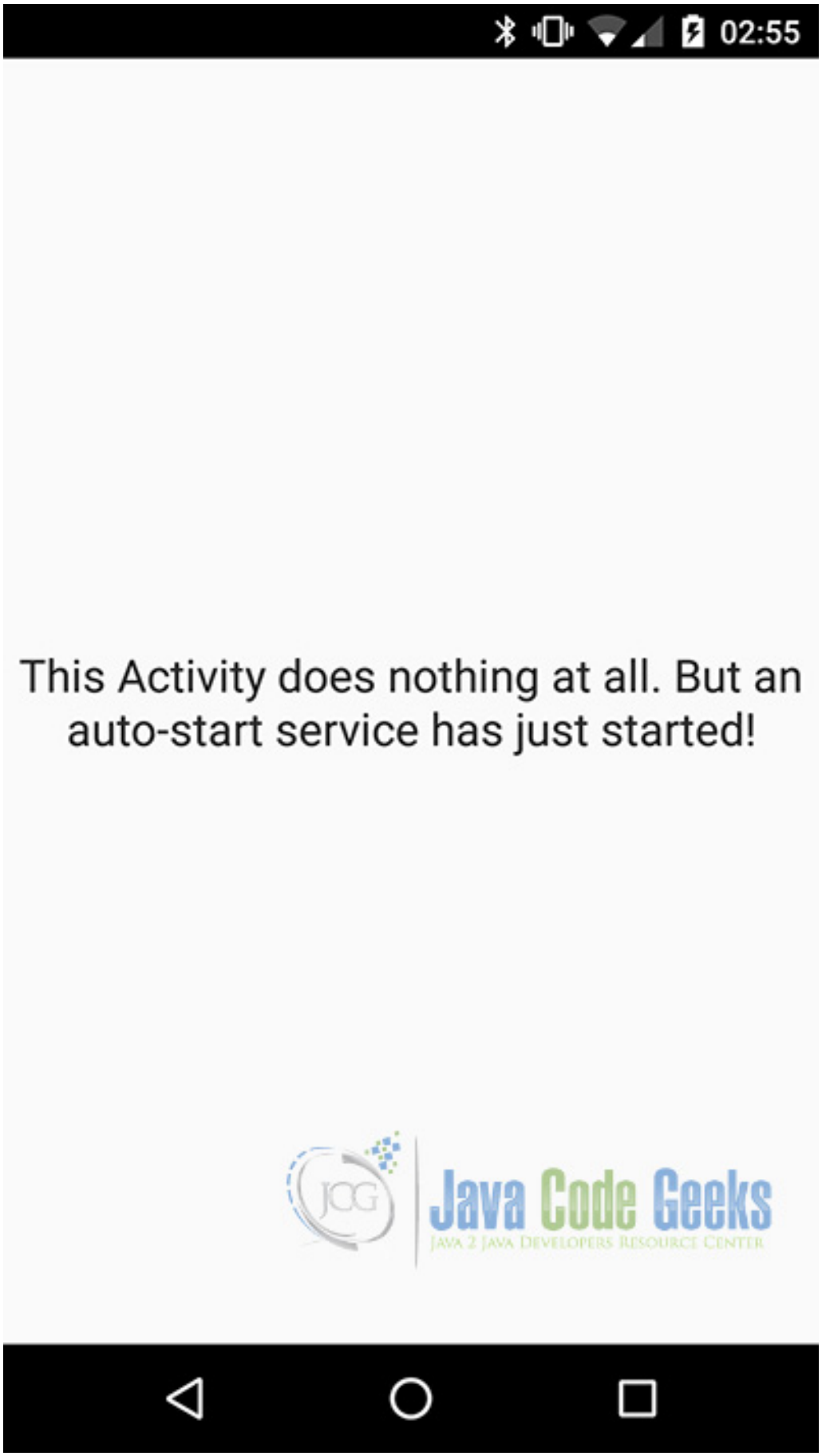

Figure 4.5: This is this how our application should look like.

This application does not do anythining in its activity. All the work is done in the service that starts to run on the boot of our device, or on update of our application.

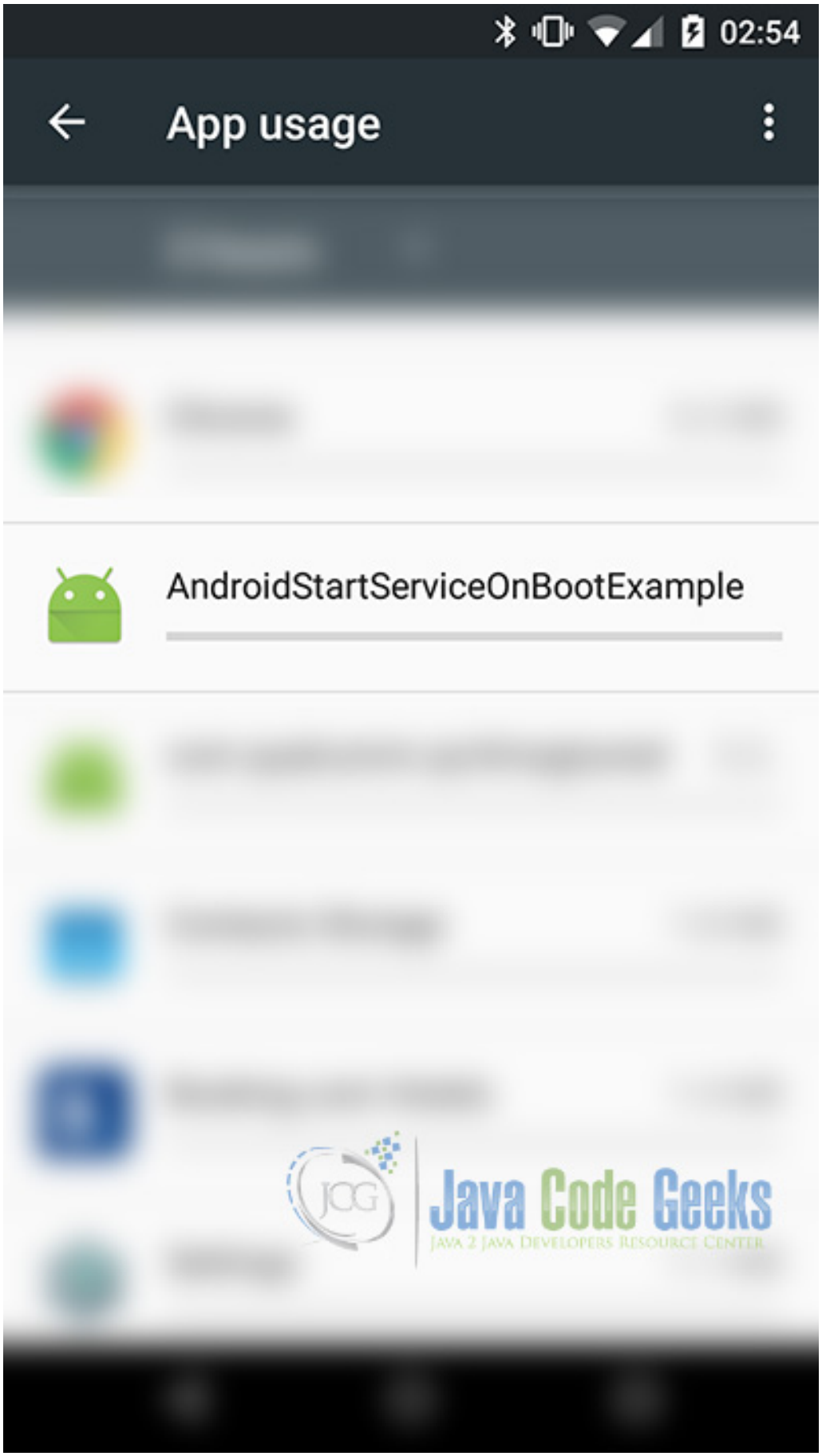

Figure 4.6: This is the service that runs in our device.

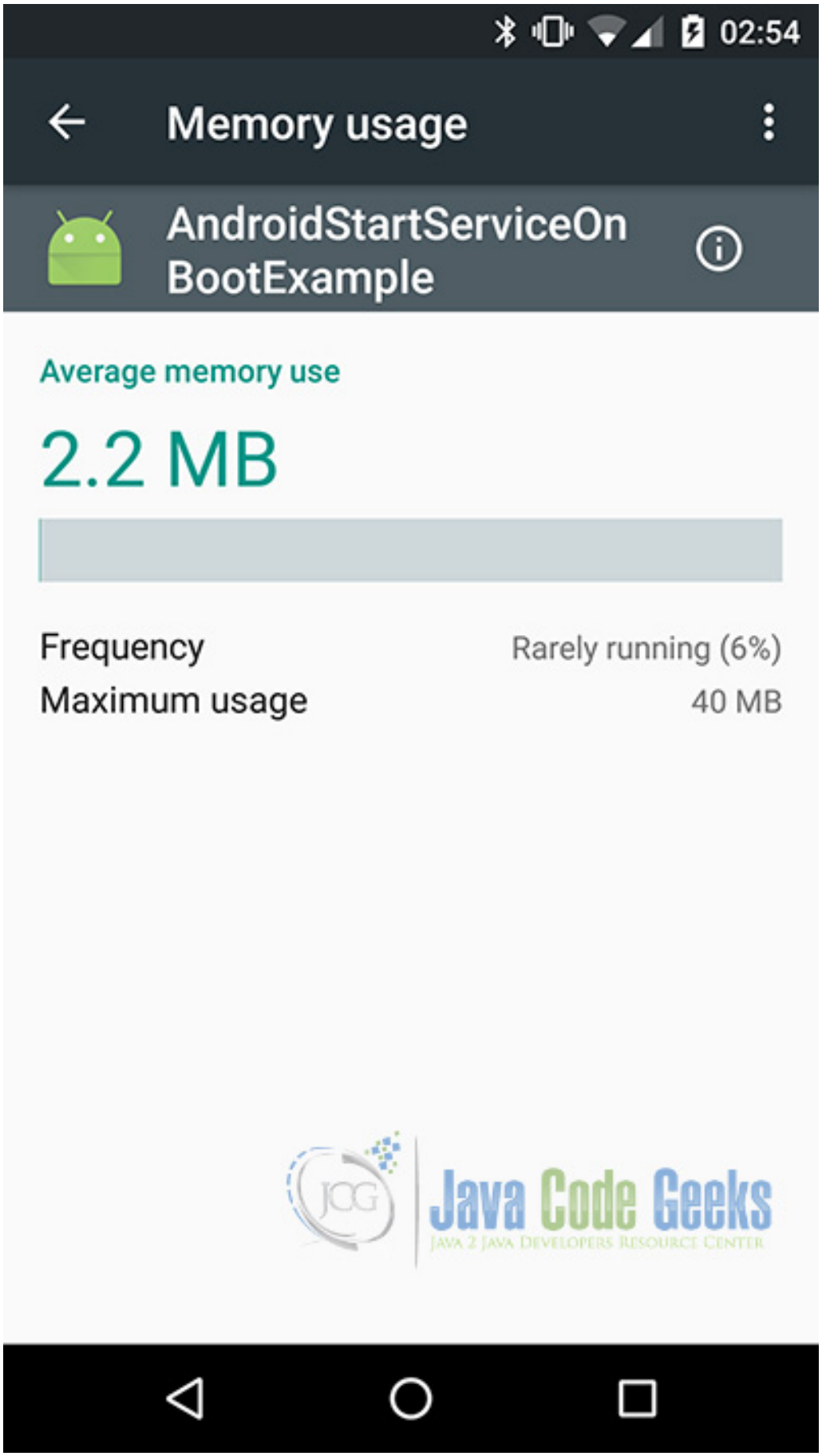

Figure 4.7: This is the service that runs in our device.

4.7 Download the Android Studio Project

This was an example of Android AndroidServiceStartOnBoot.

Download You can download the full source code of this example here: **AndroidStartServiceOnBootExample**

Chapter 5

Android Bluetooth Connection Example

The Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The Android Bluetooth APIs provide us access to the Bluetooth functionality in order to wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features. So, if we want to exchange data between different devices through our application, Bluetooth is a common way to connect two devices that support Bluetooth.

In this example we are going to create an application which activates Bluetooth, finds Bluetooth devices that may be near, scans for other undiscovered Bluetooth devices and finally uses Bluetooth connection to create a Chat Application between two devices.

5.1 Introduction

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.8
- Android Studio 2.1.2
- Android SDK 6.0

For this example we have used methods and implementation from the **Android Bluetooth Example** related to Android Bluetooth searching and pairing devices. We have also made use of the **Android RecyclerView Example** for the ui and layout of our Bluetooth Chat.

Let's take a closer look:

5.2 Create a New Android Studio Project

Open Android Studio and choose Start a new Android Studio Project in the welcome screen.

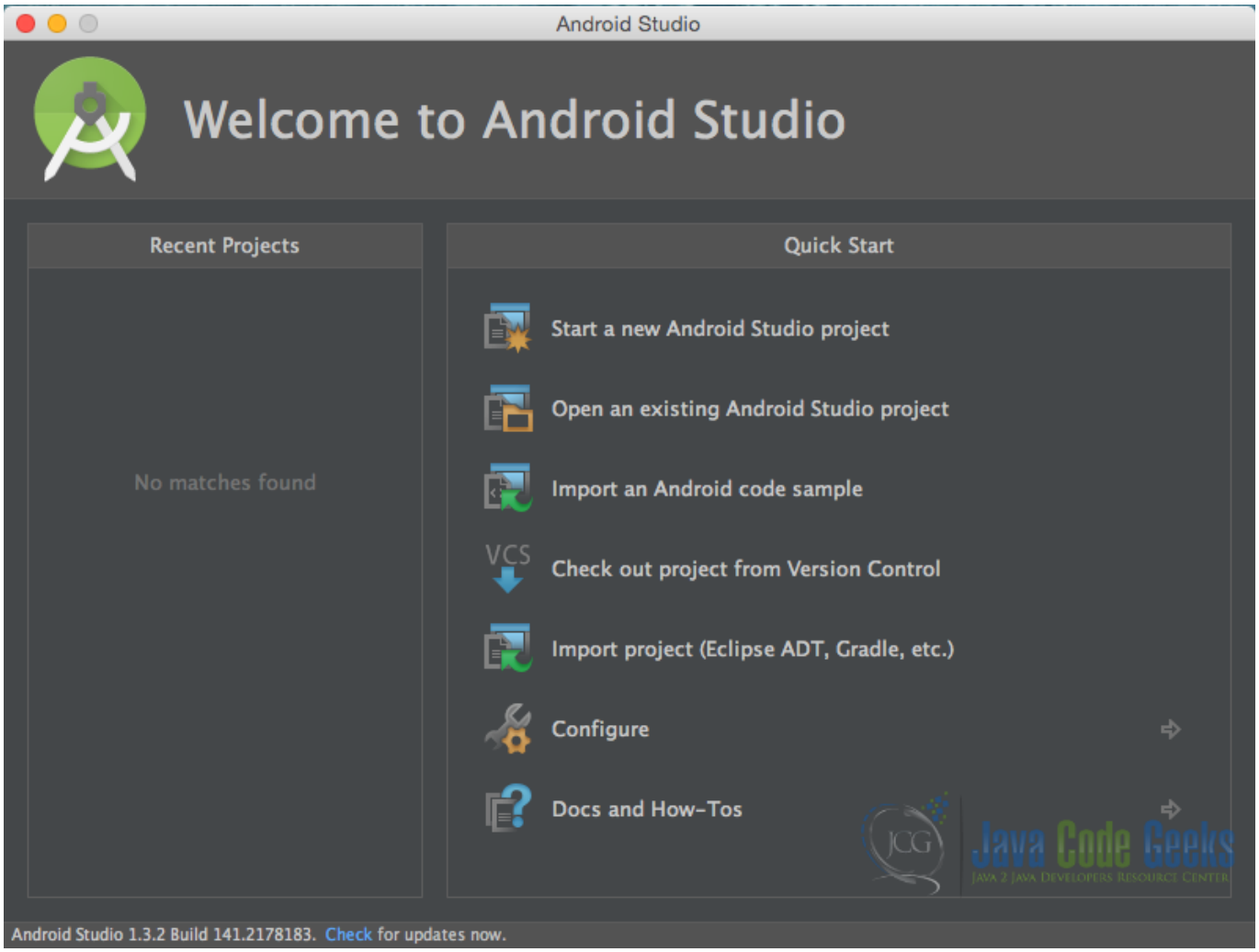

Figure 5.1: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

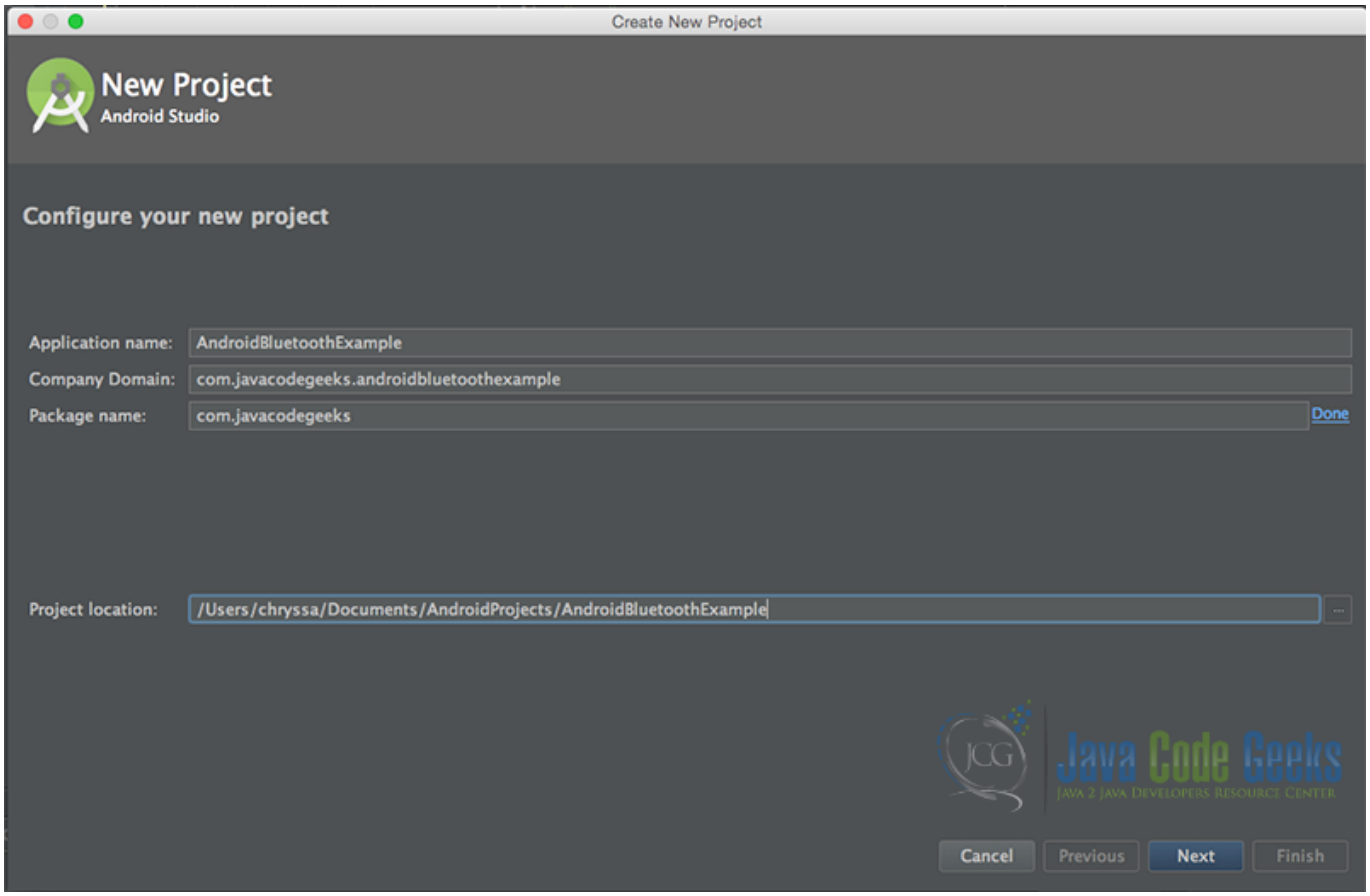

Figure 5.2: Configure your new project screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

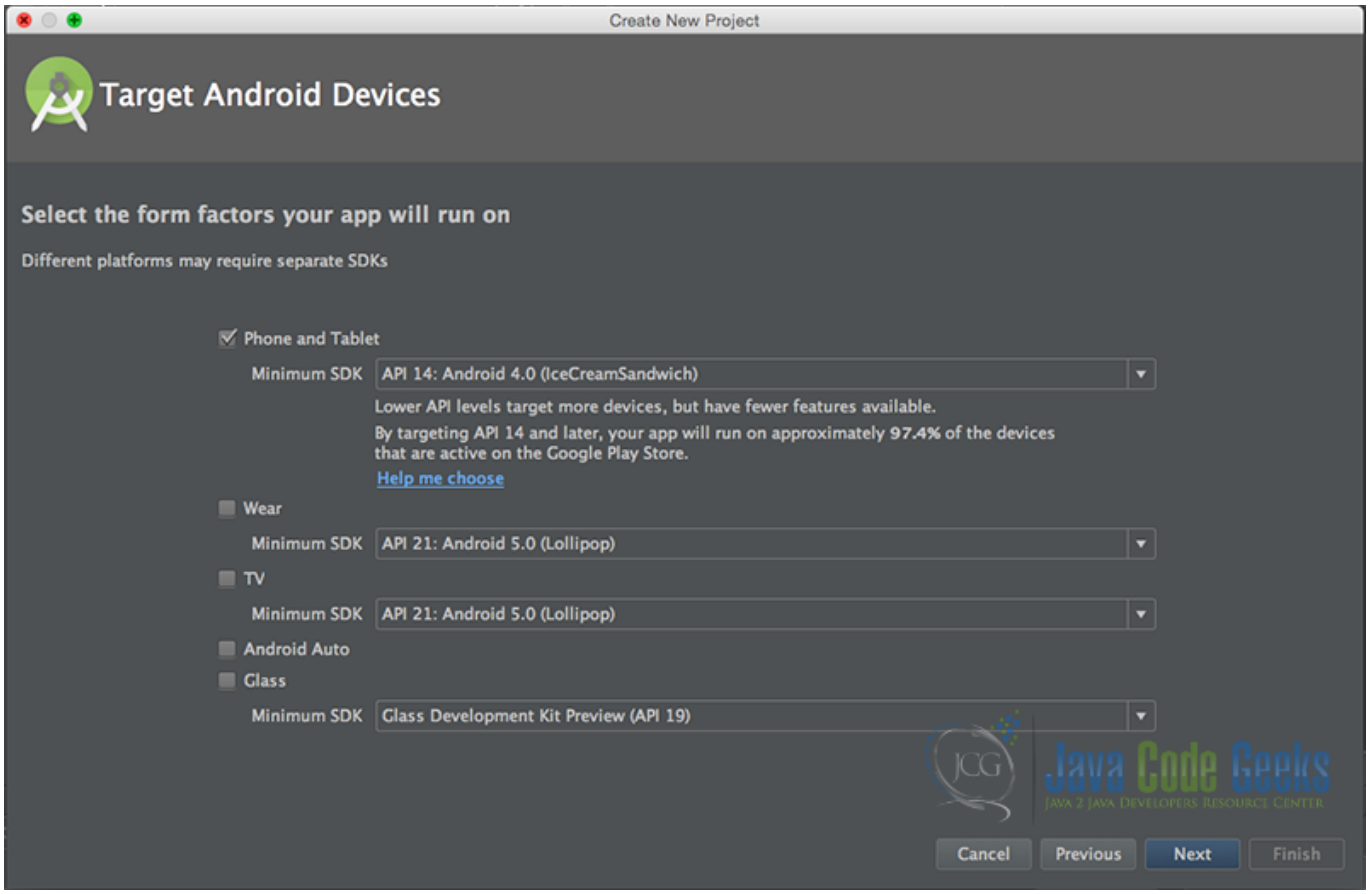

Figure 5.3: Target Android Devices screen.

In the next window you should choose to Add an activity to Mobile. In our example, we will choose to create a project with no activity, so choose: Add no activity.

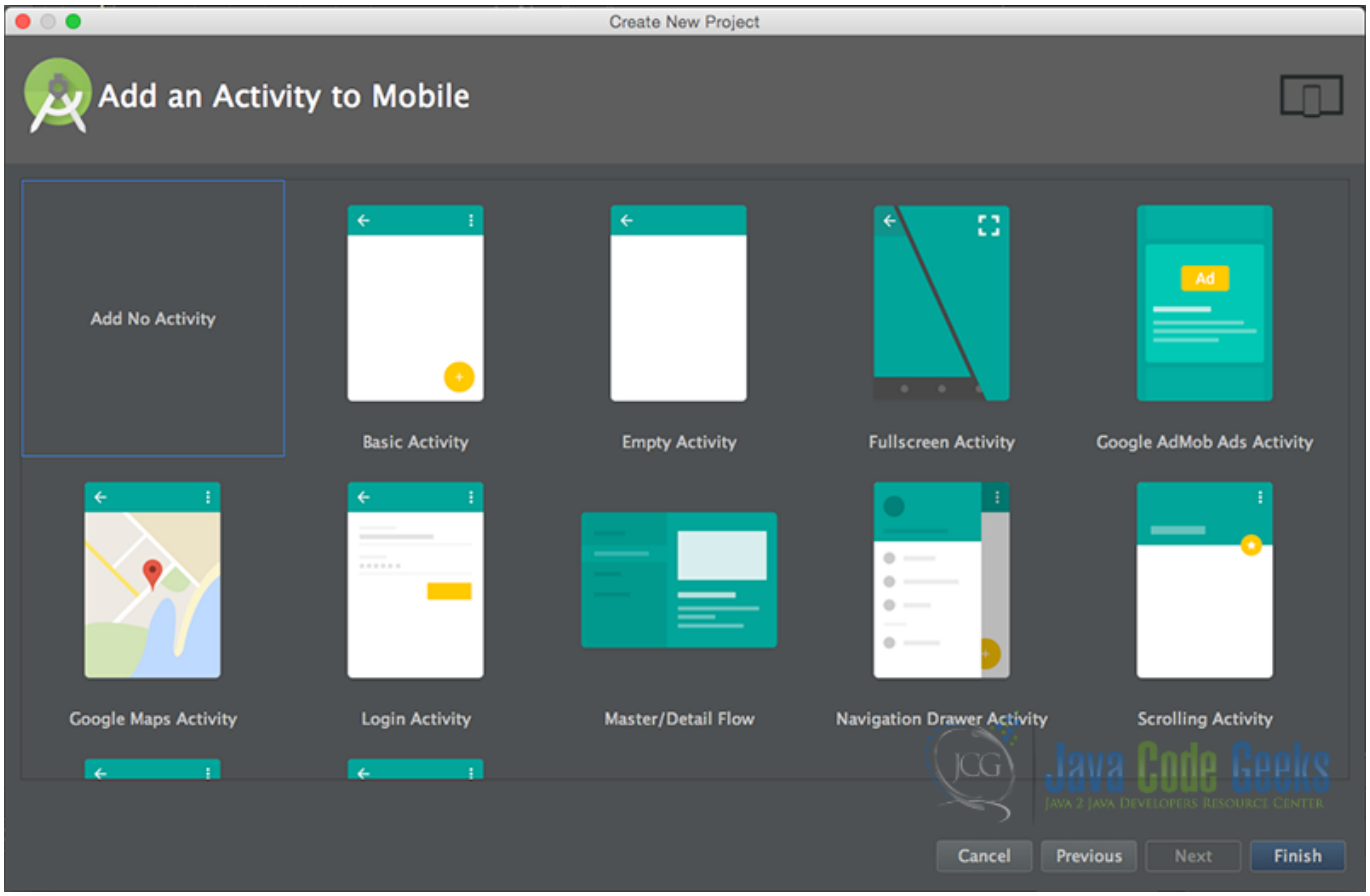

Figure 5.4: Add an activity to Mobile. Choose: Add no activity.

Now press finish, and our project has just been created!

5.3 Create the layout of the BluetoothChat

Add a new xml file inside `/res/layout` folder, with name `main_activity.xml`. We should have `/layout/main_activity.xml` file and paste the code below.

main_activity.xml

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

    <android.support.v7.widget.RecyclerView
        android:id="@+id/my_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:scrollbars="vertical" />

    <LinearLayout
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <EditText
            android:id="@+id/edit_text_out"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:layout_weight="1" />

        <Button
            android:id="@+id/button_send"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/send" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/scan"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="connect"
            android:text="@string/connect" />

        <Button
            android:id="@+id/discoverable"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="discoverable"
            android:text="@string/discoverable" />
    </LinearLayout>
</LinearLayout>
```

5.4 Create the source code of the BluetoothChat

Add a new Java class inside `src/com.javacodegeeks.androidBluetoothExample/` so that we are going to have the `src/com.javacodegeeks.androidBluetoothExample/BluetoothChat.java` file and paste the code below.

BluetoothChat.java

```
package com.javacodegeeks.androidBluetoothExample;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v7.widget.DefaultItemAnimator;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.KeyEvent;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

import com.javacodegeeks.R;

import androidRecyclerView.MessageAdapter;


public class BluetoothChat extends Activity {

    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;


    // Key names received from the BluetoothChatService Handler
    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;
    private EditText mOutEditText;
    private Button mSendButton;

    // Name of the connected device
    private String mConnectedDeviceName = null;
    // String buffer for outgoing messages
    private StringBuffer mOutStringBuffer;

    // Local Bluetooth adapter
    private BluetoothAdapter mBluetoothAdapter = null;

    // Member object for the chat services
    private BluetoothChatService mChatService = null;

    private RecyclerView mRecyclerView;
    private LinearLayoutManager mLayoutManager;
    private MessageAdapter mAdapter;

    public int counter = 0;

    private List messageList = new ArrayList();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main_activity);

        mRecyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);
        mRecyclerView.setHasFixedSize(true);
```

```
        mLayoutManager = new LinearLayoutManager(this);
        mRecyclerView.setLayoutManager(mLayoutManager);
        mAdapter = new MessageAdapter(getBaseContext(), messageList);
        mRecyclerView.setAdapter(mAdapter);
        mRecyclerView.setItemAnimator(new DefaultItemAnimator());
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // If the adapter is null, then Bluetooth is not supported
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
    }

    @Override
    public void onStart() {
        super.onStart();
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        } else {
            if (mChatService == null) setupChat();
        }
    }

    @Override
    public synchronized void onResume() {
        super.onResume();
        if (mChatService != null) {
            if (mChatService.getState() == BluetoothChatService.STATE_NONE) {
                mChatService.start();
            }
        }
    }

    private void setupChat() {
        mOutEditText = (EditText) findViewById(R.id.edit_text_out);
        mOutEditText.setOnEditorActionListener(mWriteListener);
        mSendButton = (Button) findViewById(R.id.button_send);
        mSendButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                TextView view = (TextView) findViewById(R.id.edit_text_out);
                String message = view.getText().toString();
                sendMessage(message);
            }
        });

        // Initialize the BluetoothChatService to perform bluetooth connections
        mChatService = new BluetoothChatService(this, mHandler);

        // Initialize the buffer for outgoing messages
        mOutStringBuffer = new StringBuffer("");
    }

    @Override
    public synchronized void onPause() {
        super.onPause();

    }

    @Override
```

```
    public void onStop() {
        super.onStop();

    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stop the Bluetooth chat services
        if (mChatService != null) mChatService.stop();
    }

    private void ensureDiscoverable() {
        if (mBluetoothAdapter.getScanMode() !=
                BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
            Intent discoverableIntent = new Intent(BluetoothAdapter. ↩
                ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }


    private void sendMessage(String message) {

        // Check that we're actually connected before trying anything
        if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
            Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT).show();
            return;
        }
        // Check that there's actually something to send
        if (message.length() > 0) {
            // Get the message bytes and tell the BluetoothChatService to write
            byte[] send = message.getBytes();
            mChatService.write(send);
            // Reset out string buffer to zero and clear the edit text field
            mOutStringBuffer.setLength(0);
            mOutEditText.setText(mOutStringBuffer);
        }
    }

    // The action listener for the EditText widget, to listen for the return key
    private TextView.OnEditorActionListener mWriteListener =
            new TextView.OnEditorActionListener() {
                public boolean onEditorAction(TextView view, int actionId, KeyEvent event)  ↩
                    {
                    // If the action is a key-up event on the return key, send the message
                    if (actionId == EditorInfo.IME_NULL && event.getAction() == KeyEvent. ↩
                        ACTION_UP) {
                        String message = view.getText().toString();
                        sendMessage(message);
                    }
                    return true;
                }
            };

    // The Handler that gets information back from the BluetoothChatService
    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MESSAGE_WRITE:
```

```
                    byte[] writeBuf = (byte[]) msg.obj;
                    // construct a string from the buffer
                    String writeMessage = new String(writeBuf);
                    mAdapter.notifyDataSetChanged();
                    messageList.add(new androidRecyclerView.Message(counter++, writeMessage ↩
                        , "Me"));
                    break;
                case MESSAGE_READ:
                    byte[] readBuf = (byte[]) msg.obj;
                    // construct a string from the valid bytes in the buffer
                    String readMessage = new String(readBuf, 0, msg.arg1);
                    mAdapter.notifyDataSetChanged();
                    messageList.add(new androidRecyclerView.Message(counter++, readMessage, ↩
                         mConnectedDeviceName));
                    break;
                case MESSAGE_DEVICE_NAME:
                    // save the connected device's name
                    mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
                    Toast.makeText(getApplicationContext(), "Connected to "
                            + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
                    break;
                case MESSAGE_TOAST:
                    Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
                            Toast.LENGTH_SHORT).show();
                    break;
            }
        }
    };

    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
            case REQUEST_CONNECT_DEVICE:
                // When DeviceListActivity returns with a device to connect
                if (resultCode == Activity.RESULT_OK) {
                    // Get the device MAC address
                    String address = data.getExtras().getString(DeviceListActivity. ↩
                        EXTRA_DEVICE_ADDRESS);
                    // Get the BLuetoothDevice object
                    BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
                    // Attempt to connect to the device
                    mChatService.connect(device);
                }
                break;
            case REQUEST_ENABLE_BT:
                // When the request to enable Bluetooth returns
                if (resultCode == Activity.RESULT_OK) {
                    // Bluetooth is now enabled, so set up a chat session
                    setupChat();
                } else {
                    // User did not enable Bluetooth or an error occured
                    Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast. ↩
                        LENGTH_SHORT).show();
                    finish();
                }
        }
    }

    public void connect(View v) {
        Intent serverIntent = new Intent(this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
    }
```

```
    public void discoverable(View v) {
        ensureDiscoverable();
    }
}
```

5.5 Create the source code of the BluetoothChatService

The BluetoothChatService class does all the work for setting up and managing Bluetooth connections with other devices. It has a thread that listens for incoming connections, a thread for connecting with a device, and a thread for performing data transmissions when connected.

So, add a new Java class inside `src/com.javacodegeeks.androidBluetoothExample/` so that we are going to have the `src/com.javacodegeeks.androidBluetoothExample/BluetoothChatService.java` file and paste the code below.

BluetoothChatService.java

```
package com.javacodegeeks.androidBluetoothExample;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
 * thread for performing data transmissions when connected.
 */
public class BluetoothChatService {

    // Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothChat";

    // Unique UUID for this application
    private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200 ↩
        c9a66");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for incoming connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing connection
```

```
    public static final int STATE_CONNECTED = 3;  // now connected to a remote device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     *
     * @param context The UI Activity Context
     * @param handler A Handler to send messages back to the UI Activity
     */
    public BluetoothChatService(Context context, Handler handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mHandler = handler;
    }

    /**
     * Set the current state of the chat connection
     *
     * @param state An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        mState = state;
        // Give the new state to the Handler so the UI Activity can update
        mHandler.obtainMessage(BluetoothChat.MESSAGE_STATE_CHANGE, state, -1).sendToTarget ↩
            ();
    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
        return mState;
    }

    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity onResume()
     */
    public synchronized void start() {
        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
        // Start the thread to listen on a BluetoothServerSocket
        if (mAcceptThread == null) {
            mAcceptThread = new AcceptThread();
            mAcceptThread.start();
        }
        setState(STATE_LISTEN);
    }

    /**
     * Start the ConnectThread to initiate a connection to a remote device.
     *
     * @param device The BluetoothDevice to connect
     */
    public synchronized void connect(BluetoothDevice device) {
```

```
        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }
        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
        // Start the thread to connect with the given device
        mConnectThread = new ConnectThread(device);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection
     *
     * @param socket The BluetoothSocket on which the connection was made
     * @param device The BluetoothDevice that has been connected
     */
    public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {
        // Cancel the thread that completed the connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
        // Cancel the accept thread because we only want to connect to one device
        if (mAcceptThread != null) {
            mAcceptThread.cancel();
            mAcceptThread = null;
        }
        // Start the thread to manage the connection and perform transmissions
        mConnectedThread = new ConnectedThread(socket);
        mConnectedThread.start();
        // Send the name of the connected device back to the UI Activity
        Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.DEVICE_NAME, device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);
        setState(STATE_CONNECTED);
    }

    /**
     * Stop all threads
     */
    public synchronized void stop() {
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
```

```
            mConnectedThread = null;
        }
        if (mAcceptThread != null) {
            mAcceptThread.cancel();
            mAcceptThread = null;
        }
        setState(STATE_NONE);
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     *
     * @param out The bytes to write
     * @see ConnectedThread#write(byte[])
     */
    public void write(byte[] out) {
        // Create temporary object
        ConnectedThread r;
        // Synchronize a copy of the ConnectedThread
        synchronized (this) {
            if (mState != STATE_CONNECTED) return;
            r = mConnectedThread;
        }
        // Perform the write unsynchronized
        r.write(out);
    }

    /**
     * Indicate that the connection attempt failed and notify the UI Activity.
     */
    private void connectionFailed() {
        setState(STATE_LISTEN);
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.TOAST, "Unable to connect device");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
    }

    /**
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        setState(STATE_LISTEN);
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.TOAST, "Device connection was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
    }

    /**
     * This thread runs while listening for incoming connections. It behaves
     * like a server-side client. It runs until a connection is accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;
```

```
        public AcceptThread() {
            BluetoothServerSocket tmp = null;
            // Create a new listening server socket
            try {
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
            } catch (IOException e) {
            }
            mmServerSocket = tmp;
        }

        public void run() {
            setName("AcceptThread");
            BluetoothSocket socket = null;
            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    socket = mmServerSocket.accept();
                } catch (IOException e) {
                    break;
                }
                // If a connection was accepted
                if (socket != null) {
                    synchronized (BluetoothChatService.this) {
                        switch (mState) {
                            case STATE_LISTEN:
                            case STATE_CONNECTING:
                                // Situation normal. Start the connected thread.
                                connected(socket, socket.getRemoteDevice());
                                break;
                            case STATE_NONE:
                            case STATE_CONNECTED:
                                // Either not ready or already connected. Terminate new ↩
                                    socket.
                                try {
                                    socket.close();
                                } catch (IOException e) {
                                }
                                break;
                        }
                    }
                }
            }
        }

        public void cancel() {
            try {
                mmServerSocket.close();
            } catch (IOException e) {
            }
        }
    }

    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
```

```
    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;
        // Get a BluetoothSocket for a connection with the
        // given BluetoothDevice
        try {
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
        }
        mmSocket = tmp;
    }

    public void run() {
        setName("ConnectThread");
        // Always cancel discovery because it will slow down a connection
        mAdapter.cancelDiscovery();
        // Make a connection to the BluetoothSocket
        try {
            // This is a blocking call and will only return on a
            // successful connection or an exception
            mmSocket.connect();
        } catch (IOException e) {
            connectionFailed();
            // Close the socket
            try {
                mmSocket.close();
            } catch (IOException e2) {
            }
            // Start the service over to restart listening mode
            BluetoothChatService.this.start();
            return;
        }
        // Reset the ConnectThread because we're done
        synchronized (BluetoothChatService.this) {
            mConnectThread = null;
        }
        // Start the connected thread
        connected(mmSocket, mmDevice);
    }

    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
        }
    }
}

/**
 * This thread runs during a connection with a remote device.
 * It handles all incoming and outgoing transmissions.
 */
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
```

```
            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
            }
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[1024];
            int bytes;
            // Keep listening to the InputStream while connected
            while (true) {
                try {
                    // Read from the InputStream
                    bytes = mmInStream.read(buffer);
                    // Send the obtained bytes to the UI Activity
                    mHandler.obtainMessage(BluetoothChat.MESSAGE_READ, bytes, -1, buffer)
                            .sendToTarget();
                } catch (IOException e) {
                    connectionLost();
                    break;
                }
            }
        }

        /**
         * Write to the connected OutStream.
         *
         * @param buffer The bytes to write
         */
        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);
                // Share the sent message back to the UI Activity
                mHandler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
            }
        }
    }
}
```

5.6 Create the layout of the DeviceListActivity

Add a new xml file inside `/res/layout` folder, with name `device_list.xml`. We should have `/layout/main_activity.xml` file and paste the code below.

device_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/title_paired_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#666"
        android:paddingLeft="5dp"
        android:text="@string/title_paired_devices"
        android:textColor="#fff"
        android:visibility="gone" />

    <ListView
        android:id="@+id/paired_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:stackFromBottom="true" />

    <TextView
        android:id="@+id/title_new_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#666"
        android:paddingLeft="5dp"
        android:text="@string/title_other_devices"
        android:textColor="#fff"
        android:visibility="gone" />

    <ListView
        android:id="@+id/new_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:stackFromBottom="true" />

    <Button
        android:id="@+id/button_scan"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_scan" />
</LinearLayout>
```

5.7 Create the source code of the DeviceListActivity

This Activity lists any paired devices and devices detected in the area after discovery. When a device is chosen by the user, the MAC address of the device is sent back to the parent Activity in the result Intent.

So, add a new Java class inside `src/com.javacodegeeks.androidBluetoothExample/` so that we are going to have the `src/com.javacodegeeks.androidBluetoothExample/DeviceListActivity.java` file and paste the code below.

DeviceListActivity.java

```
package com.javacodegeeks.androidBluetoothExample;

package com.javacodegeeks.androidBluetoothExample;
```

```
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

import com.javacodegeeks.R;
import java.util.Set;

public class DeviceListActivity extends Activity {
    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";
    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter mPairedDevicesArrayAdapter;
    private ArrayAdapter mNewDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Setup the window
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.device_list);
        // Set result CANCELED incase the user backs out
        setResult(Activity.RESULT_CANCELED);
        // Initialize the button to perform device discovery
        Button scanButton = (Button) findViewById(R.id.button_scan);
        scanButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                doDiscovery();
                v.setVisibility(View.GONE);
            }
        });
        // Initialize array adapters. One for already paired devices and
        // one for newly discovered devices
        mPairedDevicesArrayAdapter = new ArrayAdapter(this, R.layout.device_name);
        mNewDevicesArrayAdapter = new ArrayAdapter(this, R.layout.device_name);
        // Find and set up the ListView for paired devices
        ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
        pairedListView.setAdapter(mPairedDevicesArrayAdapter);
        pairedListView.setOnItemClickListener(mDeviceClickListener);
        // Find and set up the ListView for newly discovered devices
        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
        newDevicesListView.setOnItemClickListener(mDeviceClickListener);
        // Register for broadcasts when a device is discovered
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);
```

```
        // Register for broadcasts when discovery has finished
        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(mReceiver, filter);
        // Get the local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        // Get a set of currently paired devices
        Set pairedDevices = mBtAdapter.getBondedDevices();
        // If there are paired devices, add each one to the ArrayAdapter
        if (pairedDevices.size() > 0) {
            findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
            for (BluetoothDevice device : pairedDevices) {
                mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress ↩
                    ());
            }
        } else {
            String noDevices = getResources().getText(R.string.none_paired).toString();
            mPairedDevicesArrayAdapter.add(noDevices);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null) {
            mBtAdapter.cancelDiscovery();
        }
        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
    }

    /**
     * Start device discover with the BluetoothAdapter
     */
    private void doDiscovery() {
        // Indicate scanning in the title
        setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);
        // Turn on sub-title for new devices
        findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);
        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering()) {
            mBtAdapter.cancelDiscovery();
        }
        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
    }

    // The on-click listener for all devices in the ListViews
    private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
        public void onItemClick(AdapterView av, View v, int arg2, long arg3) {
            // Cancel discovery because it's costly and we're about to connect
            mBtAdapter.cancelDiscovery();
            // Get the device MAC address, which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);
            // Create the result Intent and include the MAC address
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
            // Set result and finish this Activity
            setResult(Activity.RESULT_OK, intent);
            finish();
```

```
        }
    };

    // The BroadcastReceiver that listens for discovered devices and
    // changes the title when discovery is finished
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            // When discovery finds a device
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice. ↩
                    EXTRA_DEVICE);
                // If it's already paired, skip it, because it's been listed already
                if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress ↩
                        ());
                }
                // When discovery is finished, change the Activity title
            } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);
                if (mNewDevicesArrayAdapter.getCount() == 0) {
                    String noDevices = getResources().getText(R.string.none_found).toString ↩
                        ();
                    mNewDevicesArrayAdapter.add(noDevices);
                }
            }
        }
    };
}
```

5.8 AndroidManifest.xml

The AndroidManifest.xml of our project includes the `BLUETOOTH` permissions:

AndroidManifest.xml

```
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
            android:name=".androidBluetoothExample.BluetoothChat"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
        </activity>

        <activity
            android:name=".androidBluetoothExample.DeviceListActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/select_device" />
    </application>
</manifest>
```

5.9 build.gradle

We should add the AppCompat V7 support support library as well as RecyclerView V7 support support library the in our project. We can add them as dependencies to our application via build.gradle file.

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.0"

    defaultConfig {
        applicationId "com.javacodegeeks"
        minSdkVersion 14
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile 'com.android.support:recyclerview-v7:24.1.1'
}
```

5.10 Build, compile and run

Tip This example should run simultaneously on **two different Android devices**, in order for the Bluetooth chat to connect and run. When we build, compile and run our project, the main BluetoothChat should look like this:

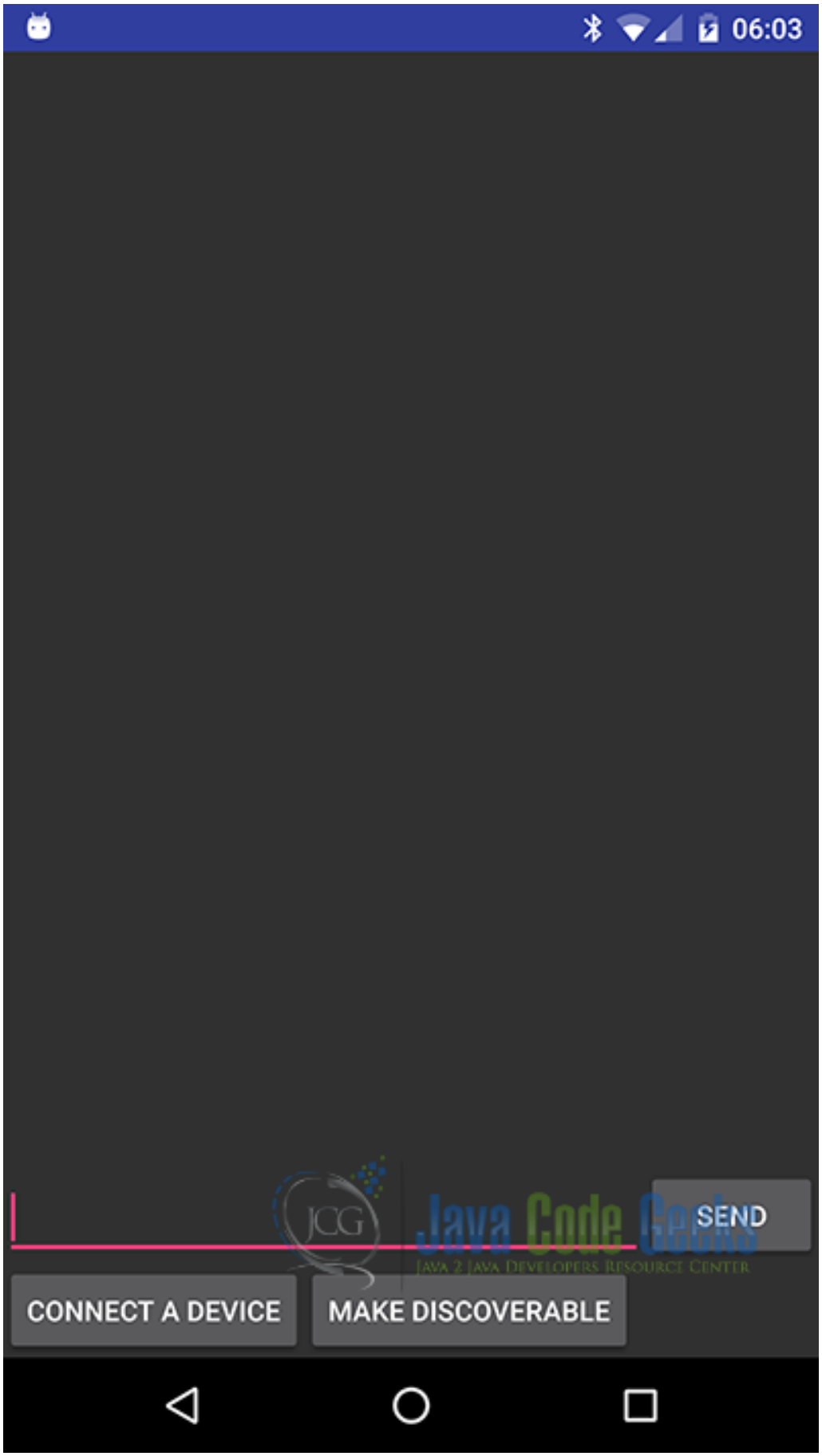

Figure 5.5: This is the first screen of the Bluetooth Connection example.

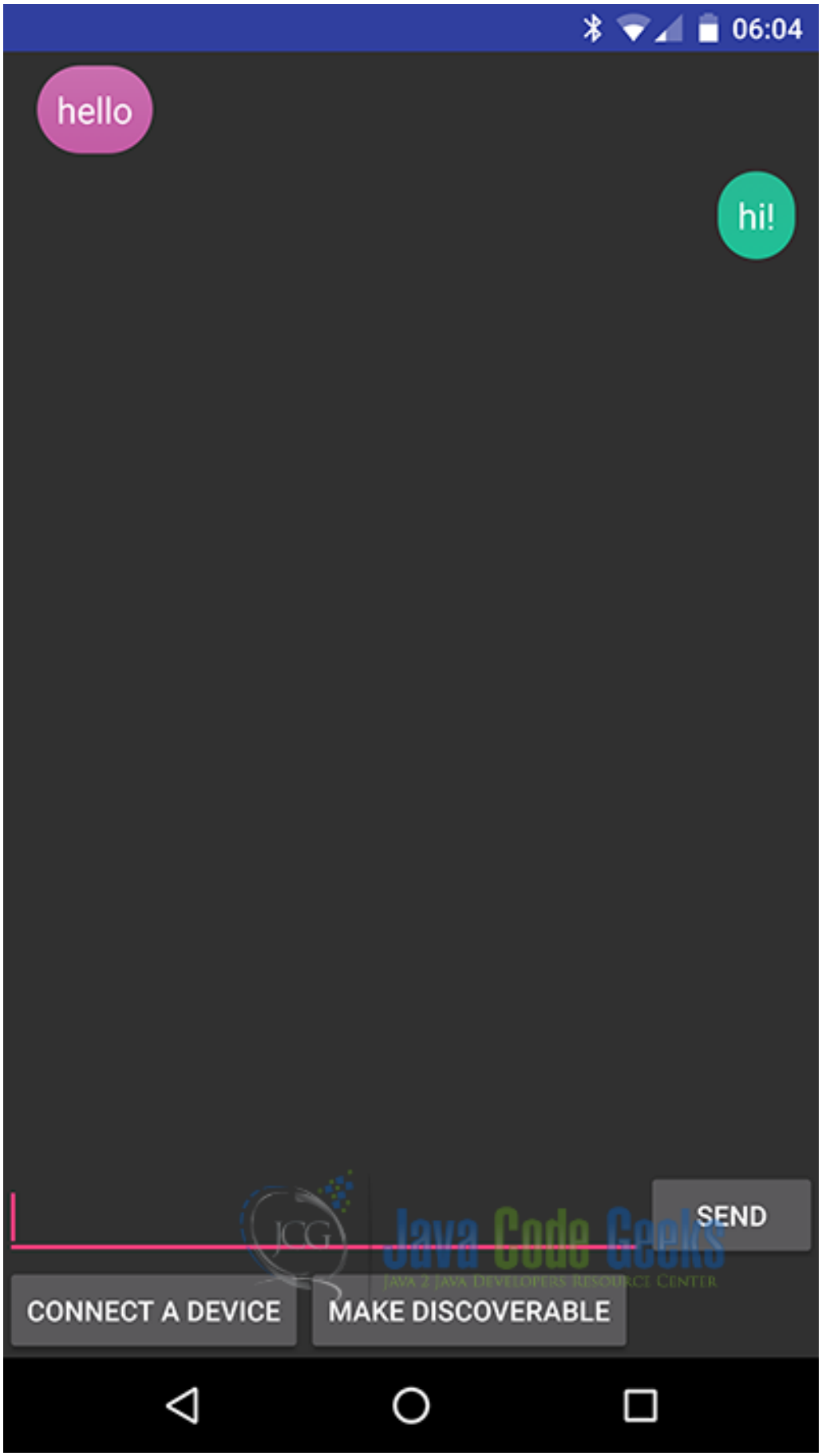

Figure 5.6: This is the BluetoothChat screen.

5.11 Download the Android Studio Project

This was an example of Android Bluetooth Connection - BluetoothChat Example.

Download You can download the full source code of this example here: **AndroidBluetoothExample.zip**

Chapter 6

Android Multitouch Example

A multi-touch gesture is when multiple pointers (fingers) touch the screen at the same time. The basic class for support touch and multitouch in Android is the MotionEvent class. Motion events describe actions in a set of axis values. The actions are the states that occur when a finger is going down or up. The axis values describe the position of this event and other movement properties.

There are single touch events and multi touch events. It depends on the screen of the device if it can handle and report multiple touch events. Multi-touch screens emit one movement trace for each finger, meaning for each pointer.

Every pointer (every finger touch) has a unique id, that is assigned when it first touches the screen. A pointer id remains valid until the pointer eventually goes up.

In order for our views to react to touch events in an Activity, an OnTouchListener should be registered for each of the views.

So, In this example we are going to extend a FrameLayout and use the multi touch events in order to give this Layout some special behaviour. We are going to make a custom onTouchListener for this FrameLayout and override the `dispatchTouchEvent()`.

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Android Studio 1.3.2
- Android SDK 5.1

Let's take a closer look:

6.1 Create a New Android Studio Project

Open Android Studio and choose Start a new Android Studio Project in the welcome screen.

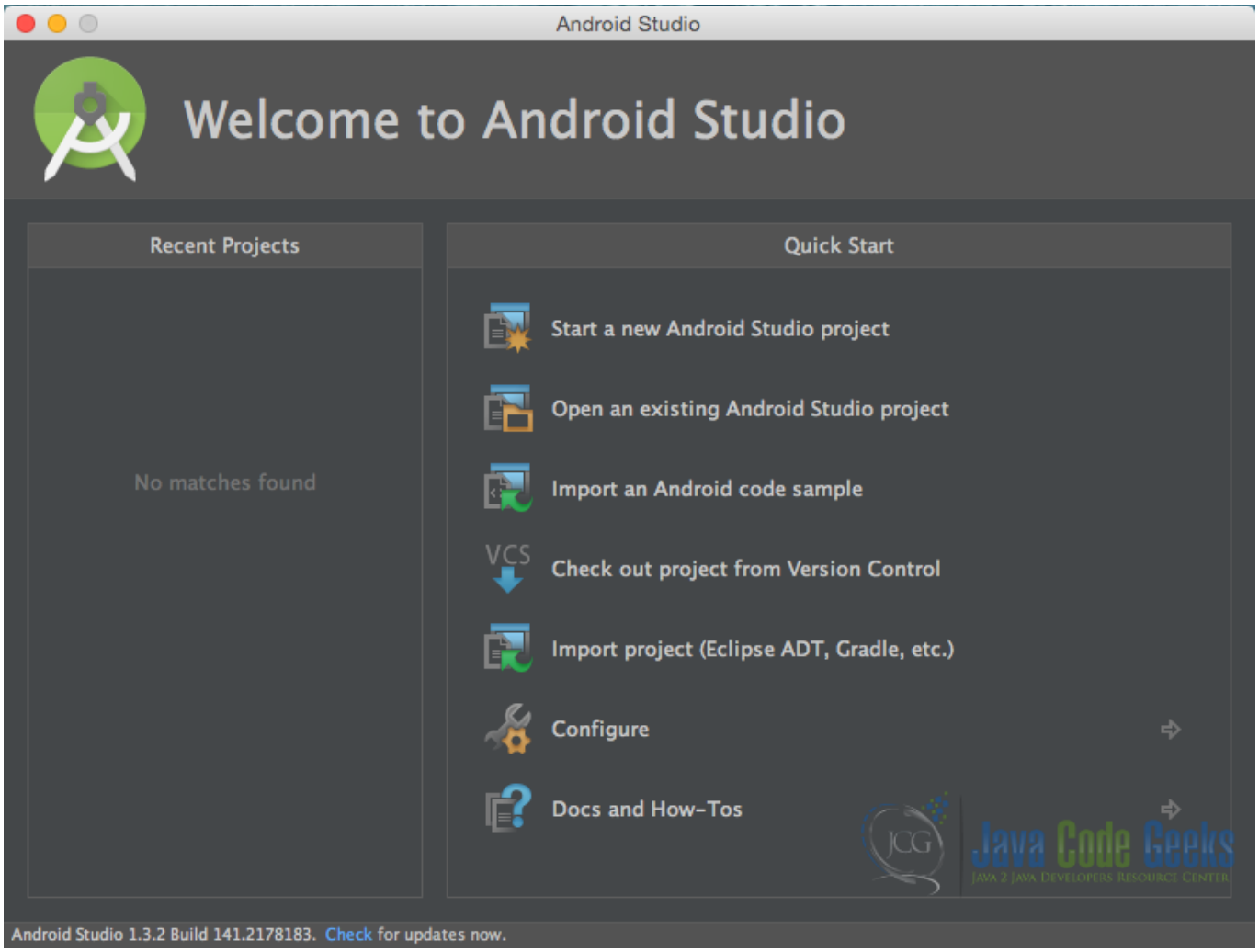

Figure 6.1: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

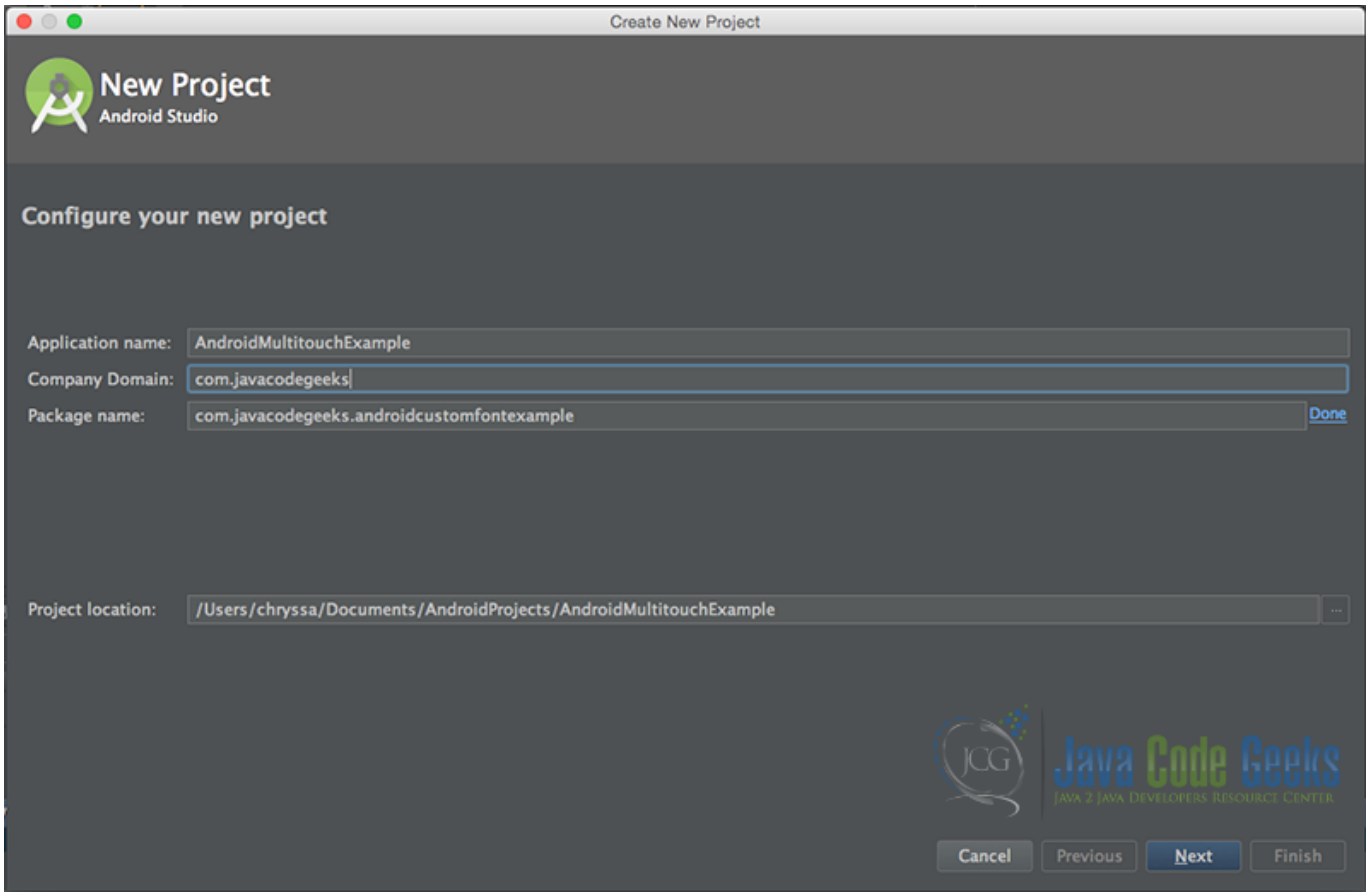

Figure 6.2: Configure your new project screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

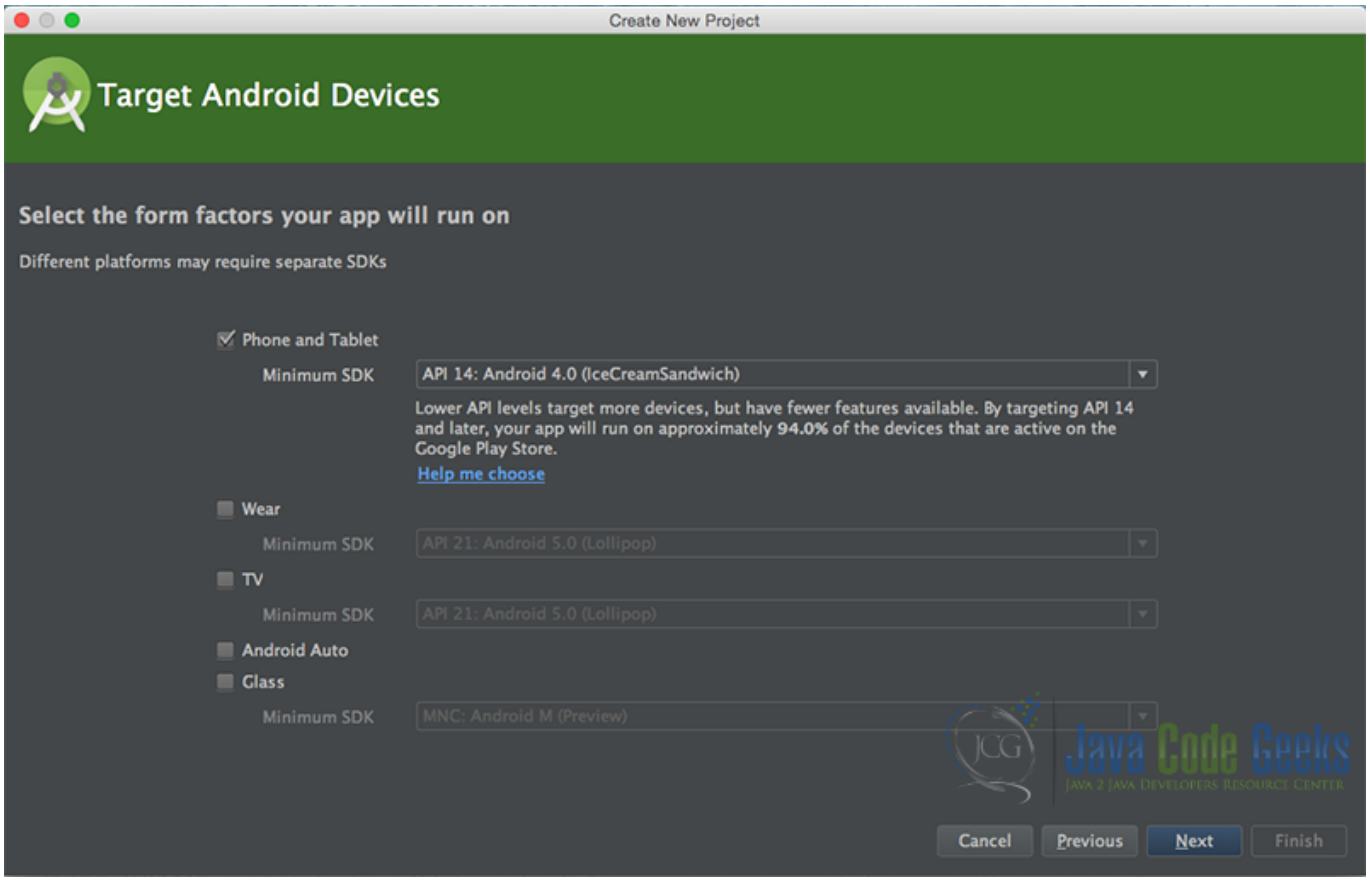

Figure 6.3: Target Android Devices screen.

In the next window you should choose to Add an activity to Mobile. In our example, we will choose to create a project with no activity, so choose: Add no activity.

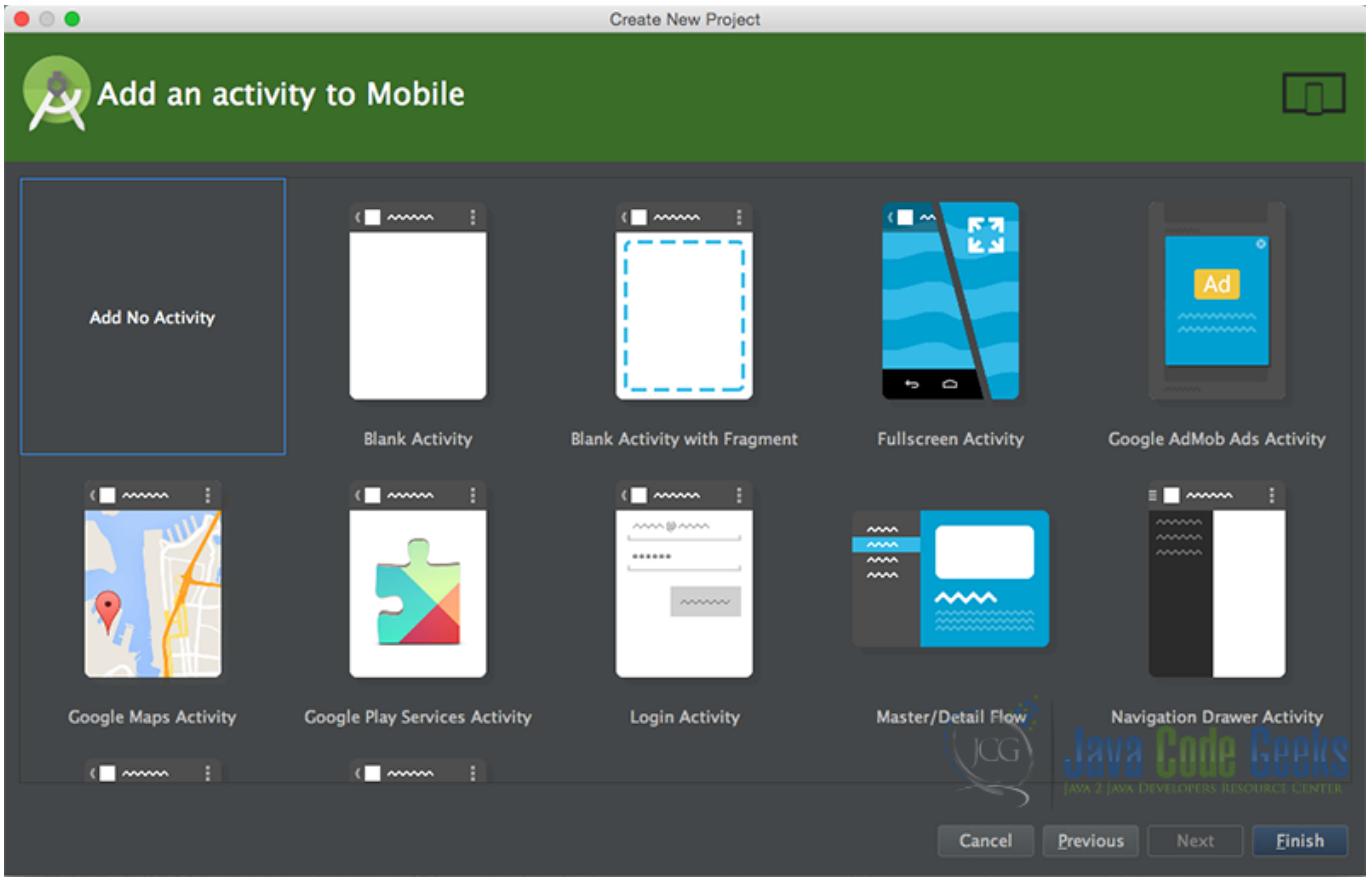

Figure 6.4: Add an activity to Mobile. Choose: Add no activity.

Now press finish, and our project has just been created.

6.2 Create the layout of the project

Add a new xml file inside `/res/layout folder`, with name `main_activity.xml`. We should have the `res/layout/main_activity.xml` file and paste the code below.

main_activity.xml

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

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="58dp"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:title="@string/app_name"></android.support.v7.widget.Toolbar>

    <com.javacodegeeks.androidmultitouchexample.TouchableFrameLayout
        android:id="@+id/touchable_frame"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/status"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:text="@string/noActionKey"
            android:textColor="#262626"
            android:textSize="30dp" />

    </com.javacodegeeks.androidmultitouchexample.TouchableFrameLayout>
</LinearLayout>
```

6.3 Creating the source code of the TouchableFrameLayout FrameLayout

Add a new Java class inside `src/com.javacodegeeks.androidmultitouchexample/` so that we are going to have the `src/com.javacodegeeks.androidmultitouchexample/TouchableFrameLayout.java` file and paste the code below.

TouchableFrameLayout.java

```
package com.javacodegeeks.androidmultitouchexample;

import android.content.Context;
import android.graphics.PointF;
import android.util.AttributeSet;
import android.util.SparseArray;
import android.view.MotionEvent;
import android.widget.FrameLayout;

public class TouchableFrameLayout extends FrameLayout {

    private SparseArray mActivePointers = new SparseArray();

    private OnTouchListener onTouchListener;

    float lastXPosition;
    float lastYPosition;
    double lastdist = 0;

    Context ctx;
    private boolean onScaleMove = false;

    public TouchableFrameLayout(Context context) {
        super(context);
        ctx = context;
    }


    public TouchableFrameLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        ctx = context;
    }

    public TouchableFrameLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        ctx = context;
    }
```

```
public void setTouchListener(OnTouchListener onTouchListener) {
    this.onTouchListener = onTouchListener;
}

@Override
public boolean dispatchTouchEvent(final MotionEvent event) {
    if (onTouchListener == null)
        return false;

    int pointerIndex = event.getActionIndex();
    int pointerId = event.getPointerId(pointerIndex);

    switch (event.getAction() & MotionEvent.ACTION_MASK) {
        case MotionEvent.ACTION_DOWN:

            lastYPosition = event.getY();
            lastXPosition = event.getX();
            onTouchListener.onTouch();
            return true;

        case MotionEvent.ACTION_UP:
            onScaleMove = false;
            lastdist = 0;
            onTouchListener.onRelease();
            break;

        case MotionEvent.ACTION_MOVE:
            int diffY = (int) (event.getY() - lastYPosition);
            int diffX = (int) (event.getX() - lastXPosition);

            lastYPosition = event.getY();
            lastXPosition = event.getX();

            //Check if the action was jitter
            if (Math.abs(diffX) > 4 || Math.abs(diffY) > 4) {

                if (onScaleMove) {
                    double dist = 0;

                    if (event.getPointerCount() >= 2) {
                        dist = Math.sqrt(Math.pow(event.getX(0) - event.getX(1), 2) + ↩
                            Math.pow(event.getY(0) - event.getY(1), 2));
                    }

                    if ((Math.abs(dist - lastdist) > 10) && (lastdist > 0) && (dist > ↩
                        0)) {
                        if (dist < lastdist) {
                            onTouchListener.onPinchIn();
                        } else if (dist == lastdist) {
                            //    onTouchListener.onPinchStable();
                        } else {
                            onTouchListener.onPinchOut();
                        }
                    } else {
                        onTouchListener.onTwoFingersDrag();
                    }

                    lastdist = dist;
                    return false;
                } else {
                    onTouchListener.onMove();
```

```
                    }

                }
                break;
            case MotionEvent.ACTION_CANCEL: {
                onScaleMove = false;
                mActivePointers.remove(pointerId);
                onTouchListener.onRelease();
                break;
            }
            case MotionEvent.ACTION_POINTER_DOWN:
                onScaleMove = true;
                onTouchListener.onSecondFingerOnLayout();
                PointF f = new PointF();
                f.x = event.getX(pointerIndex);
                f.y = event.getY(pointerIndex);
                mActivePointers.put(pointerId, f);

                return false;
        }
        return super.dispatchTouchEvent(event);
    }


    public interface OnTouchListener {
        void onTouch();

        void onRelease();

        void onPinchIn();

        void onPinchOut();

        void onMove();

        void onTwoFingersDrag();

        void onSecondFingerOnLayout();
    }
}
```

Let's see in detail the code above.

We have made a custom OnTouchListener interface that our custom layout is going to implement.

```
public interface OnTouchListener {
        void onTouch();

        void onRelease();

        void onPinchIn();

        void onPinchOut();

        void onMove();

        void onTwoFingersDrag();

        void onSecondFingerOnLayout();
    }
```

```
public void setTouchListener(OnTouchListener onTouchListener) {
```

```
        this.onTouchListener = onTouchListener;
    }
```

So, our custom layout should control when to dispatch the events above. For this reason we have to override the `dispatchTouchEvent(final MotionEvent event)`.

```
case MotionEvent.ACTION_DOWN:
                lastYPosition = event.getY();
                lastXPosition = event.getX();
                onTouchListener.onTouch();
                return true;
```

In this snippet, the first pointer that touches the screen. With `event.getY();` and `event.getX();` we can get the co-ordinates of the X and Y axis of this event.

```
case MotionEvent.ACTION_POINTER_DOWN:
                onScaleMove = true;
                onTouchListener.onSecondFingerOnLayout();
                PointF f = new PointF();
                f.x = event.getX(pointerIndex);
                f.y = event.getY(pointerIndex);
                mActivePointers.put(pointerId, f);

                return false;
```

With the code above, we can get the second pointer the first pointer co-ordinates of the X and Y axis.

```
if (event.getPointerCount() >= 2) {
        dist = Math.sqrt(Math.pow(event.getX(0) - event.getX(1), 2) + Math.pow(event.getY ↩
            (0) - event.getY(1), 2));
}

if ((Math.abs(dist - lastdist) > 10) && (lastdist > 0) && (dist > 0)) {
        if (dist < lastdist) {
                onTouchListener.onPinchIn();
        } else if (dist == lastdist) {
                //    onTouchListener.onPinchStable();
        } else {
                onTouchListener.onPinchOut();
        }
} else {
        onTouchListener.onTwoFingersDrag();
}
```

And with this piece of code, we can get distance between the two finger pointers and figure out if its is a "pinch" meaning a scale between the two fingers, or a simple drag.

6.4 Creating the source code of the main AndroidMultitouchActivity Activity

Add a new Java class inside `src/com.javacodegeeks.androidmultitouchexample/` so that we are going to have the `src/com.javacodegeeks.androidmultitouchexample/AndroidMultitouchActivity.java` file and paste the code below.

AndroidMultitouchActivity.java

```
package com.javacodegeeks.androidmultitouchexample;

import android.app.Activity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
```

```
import android.widget.TextView;

public class AndroidMultitouchActivity extends Activity {

    private Toolbar toolbar;

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

        toolbar = (Toolbar) findViewById(R.id.toolbar);
        toolbar.setTitle(R.string.app_name);

        final TextView status = (TextView) findViewById(R.id.status);

        TouchableFrameLayout frame = (TouchableFrameLayout) findViewById(R.id. ↩
            touchable_frame);
        frame.setTouchListener(new TouchableFrameLayout.OnTouchListener() {
            @Override
            public void onTouch() {
                status.setText(R.string.onTouchKey);
            }

            @Override
            public void onRelease() {
                status.setText(R.string.onReleaseKey);
            }

            @Override
            public void onPinchIn() {
                status.setText(R.string.onPinchInKey);
            }

            @Override
            public void onPinchOut() {
                status.setText(R.string.onPinchOutKey);
            }

            @Override
            public void onMove() {
                status.setText(R.string.onMoveKey);
            }

            @Override
            public void onTwoFingersDrag() {
                status.setText(R.string.onTwoFingersDragKey);
            }

            @Override
            public void onSecondFingerOnLayout() {
                status.setText(R.string.onSecondFingerOnLayout);
            }

        });
    }
}
```

6.5 Create the strings.xml

Add a new xml file inside `/res/values folder`, with name `strings.xml`. We should have the `res/values/strings.xml` file and paste the code below.

strings.xml

```
<resources>
    <string name="app_name">AndroidMultitouchExample</string>
    <string name="noActionKey">Touch me!</string>
    <string name="onTouchKey">On Touch</string>
    <string name="onMoveKey">On Move</string>
    <string name="onReleaseKey">On Release</string>
    <string name="onPinchInKey">On Pinch In</string>
    <string name="onPinchOutKey">On Pinch Out</string>
    <string name="onSecondFingerOnLayout">On Second Finger Touch</string>
    <string name="onTwoFingersDragKey">On Two Fingers Drag</string>
</resources
>
```

6.6 Android Manifest

The AndroidManifest.xml of our project is simple and contains no special permissions:

AndroidManifest.xml

```
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.androidmultitouchexample">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.javacodegeeks.androidmultitouchexample. ↩
                AndroidMultitouchActivity"
            android:label="@string/app_name">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
    </application>

</manifest>
```

6.7 build.gradle

The build.gradle of our project is simple and contains no special permissions:

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.javacodegeeks.androidmultitouchexample"
        minSdkVersion 14
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.2.1'
}
```

6.8 Build, compile and run

When we build, compile and run our project, the main AndroidMultitouchExample should look like this:

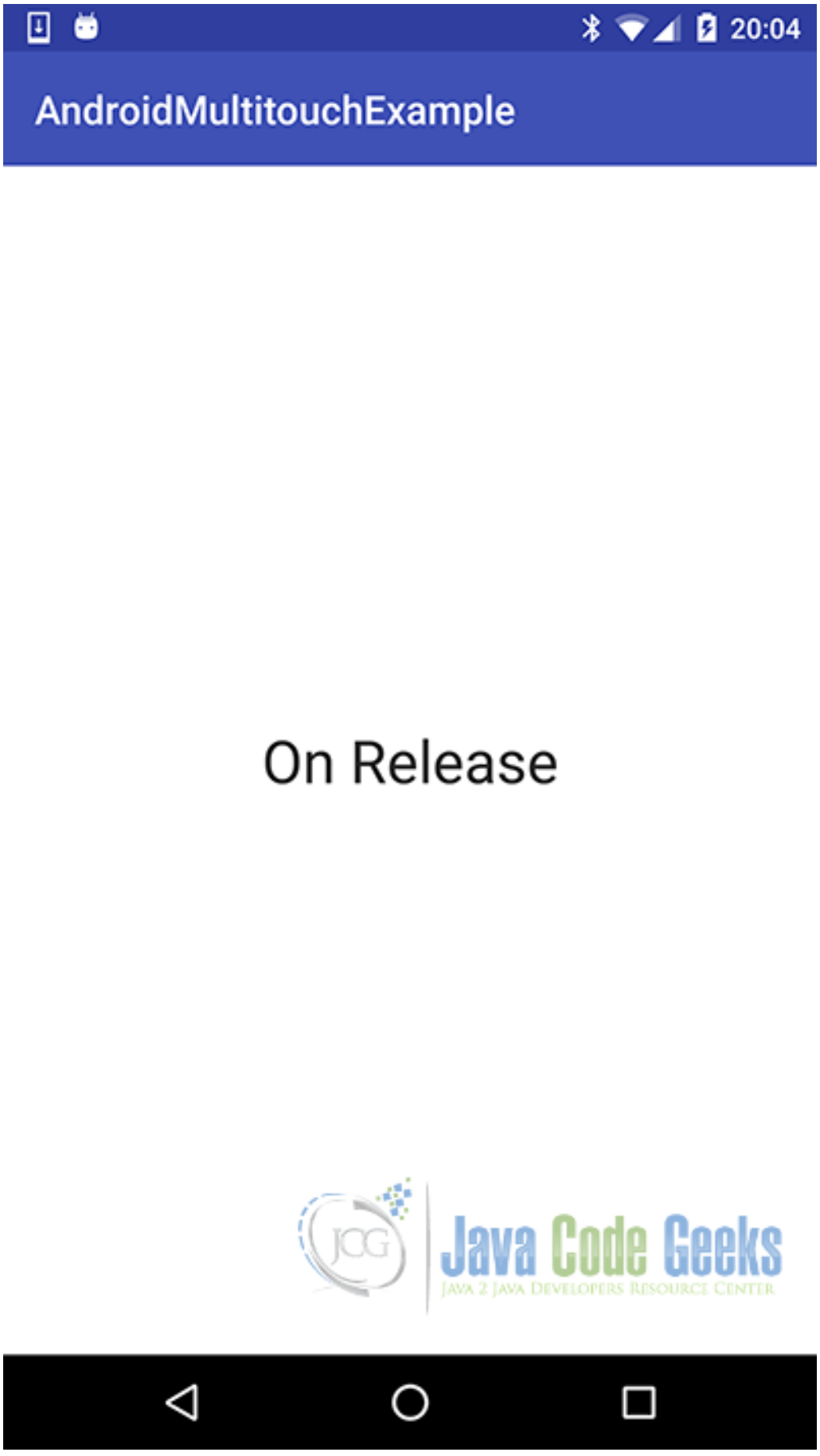

Figure 6.5: This is how our application looks.

6.9 Download the Android Studio Project

This was an example of Android Multitouch Example.

Download You can download the full source code of this example here: **AndroidMultitouchExample**

Chapter 7

Android StackView Example

The Honeycomb Android version, introduced some interesting widgets with collections. One of them is the Android StackView, a stacked card view where the front view-item can be flipped to give room for the item after it. The StackView collection may be found in several widgets, because of its view behaviour.

Generally, StackView is an AdapterView thus working with StackView is not significantly different than is working with any other AdapterView. You create an Adapter defining the contents (in this case, defining the cards), you attach the Adapter to the StackView, and put the StackView somewhere on the screen. However StackView seems to work best with children that have explicit sizes.

In this example we are going to see how to design and implement a simple Android StackView. For this example we are using the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Android Studio 2.1.2
- Android SDK 6.0

Let's take a closer look:

7.1 Create a New Android Studio Project

Open Android Studio and choose "Start a new Android Studio Project" in the welcome screen.

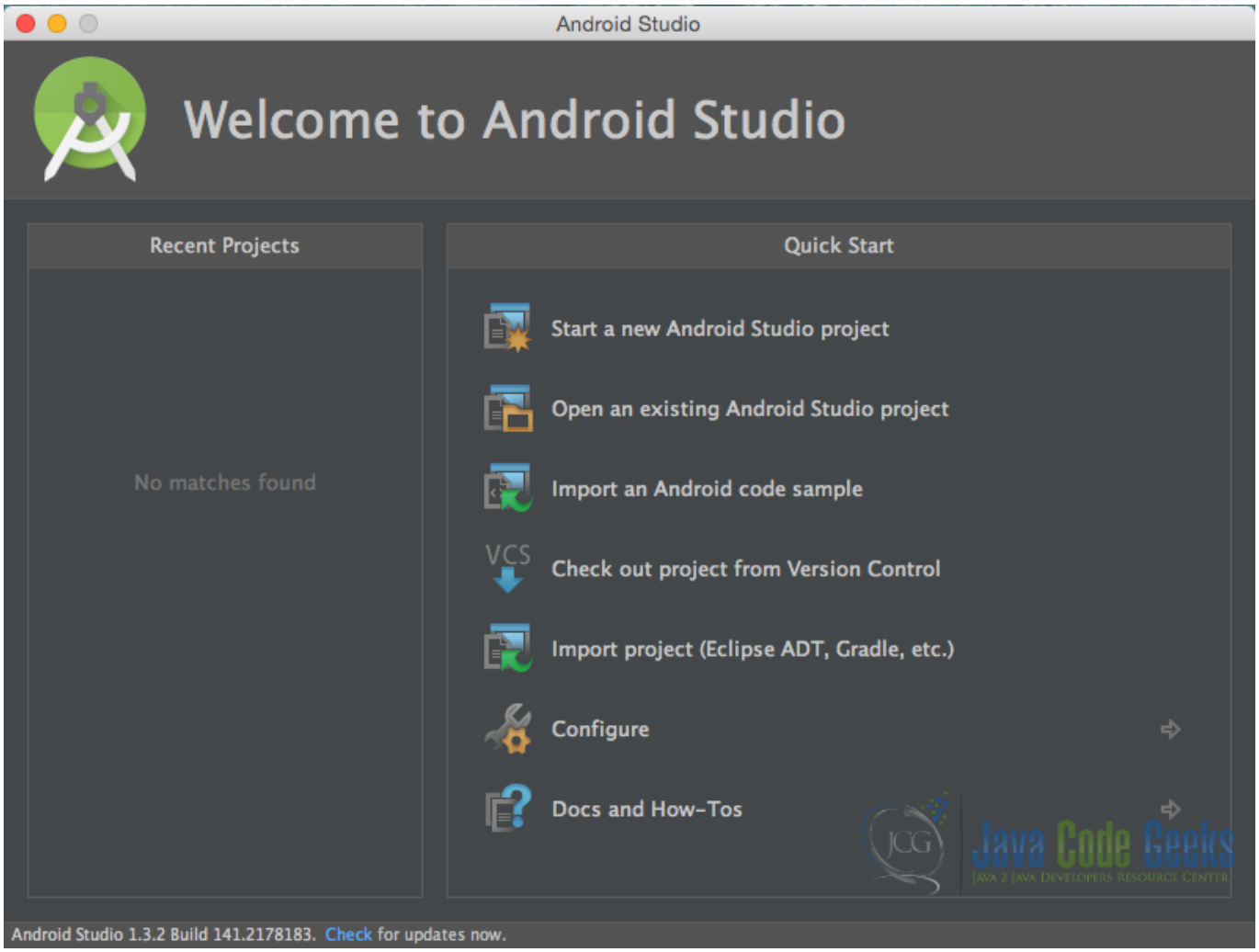

Figure 7.1: “Welcome to Android Studio” screen. Choose “Start a new Android Studio Project”.

Specify the name of the application, the project and the package.

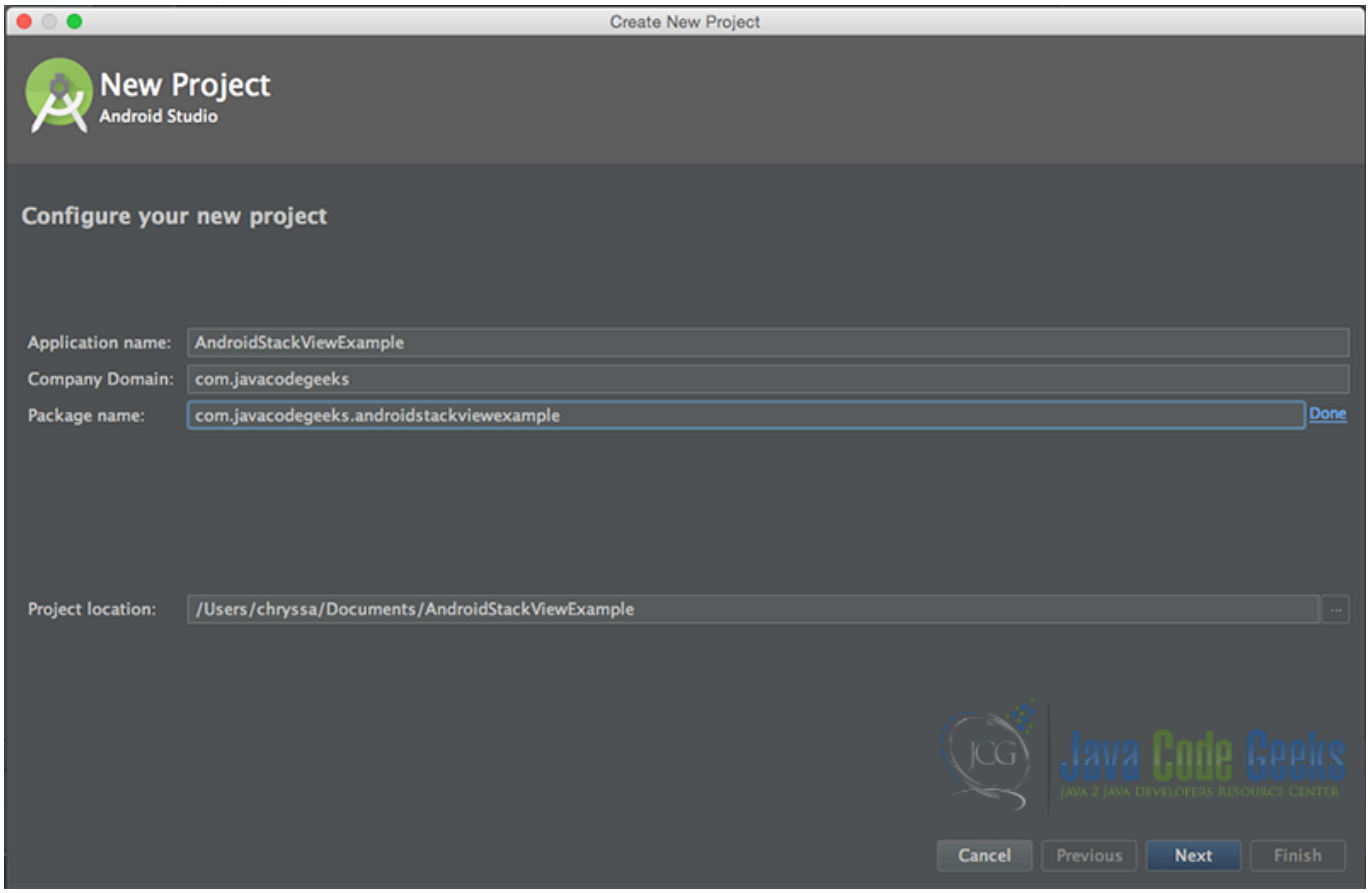

Figure 7.2: “Configure your new project” screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

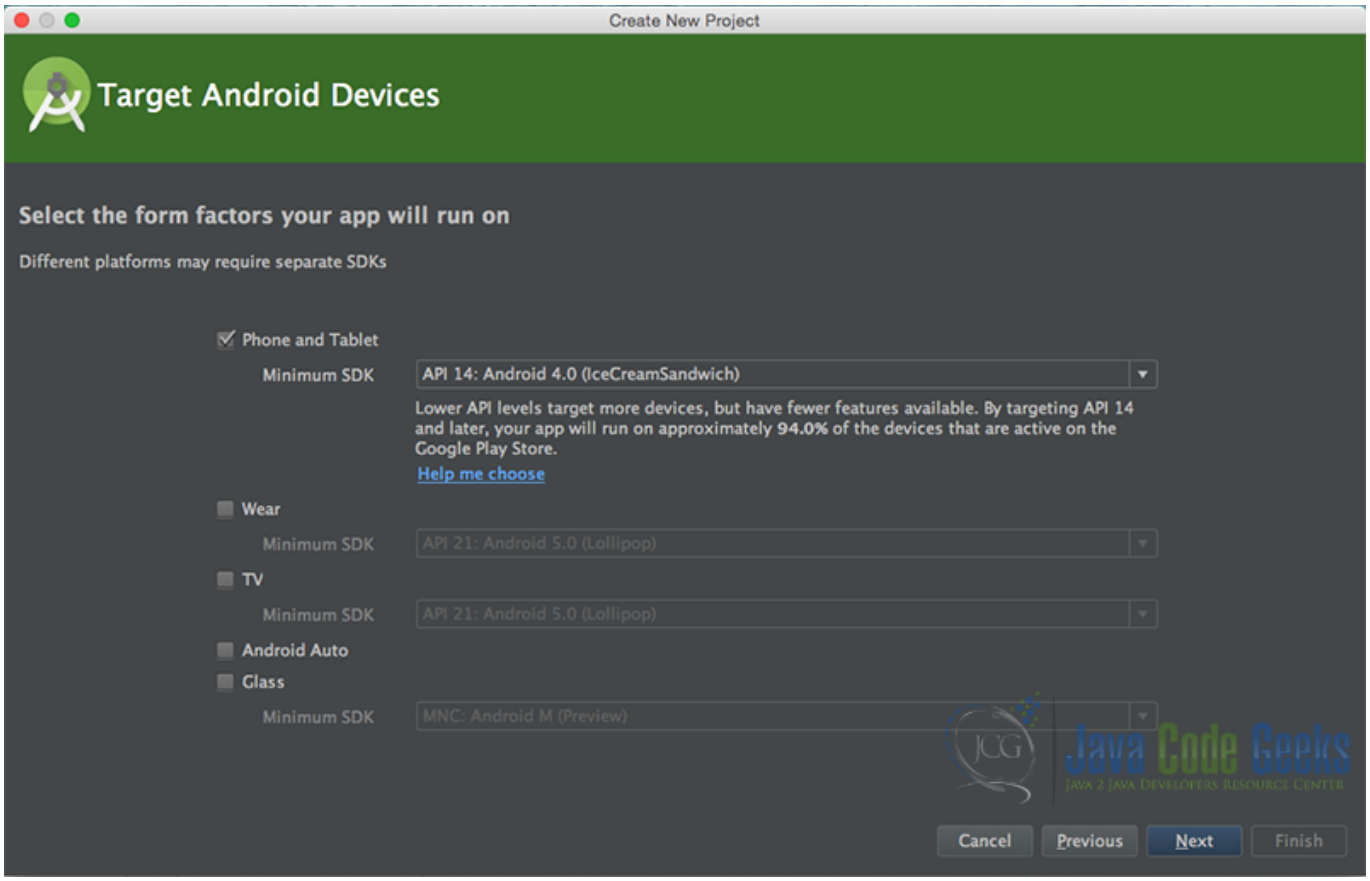

Figure 7.3: “Target Android Devices” screen.

In the next window you should choose to “Add an activity to Mobile”. In our example, we will choose to create a project with no activity, so choose: “Add no activity”.

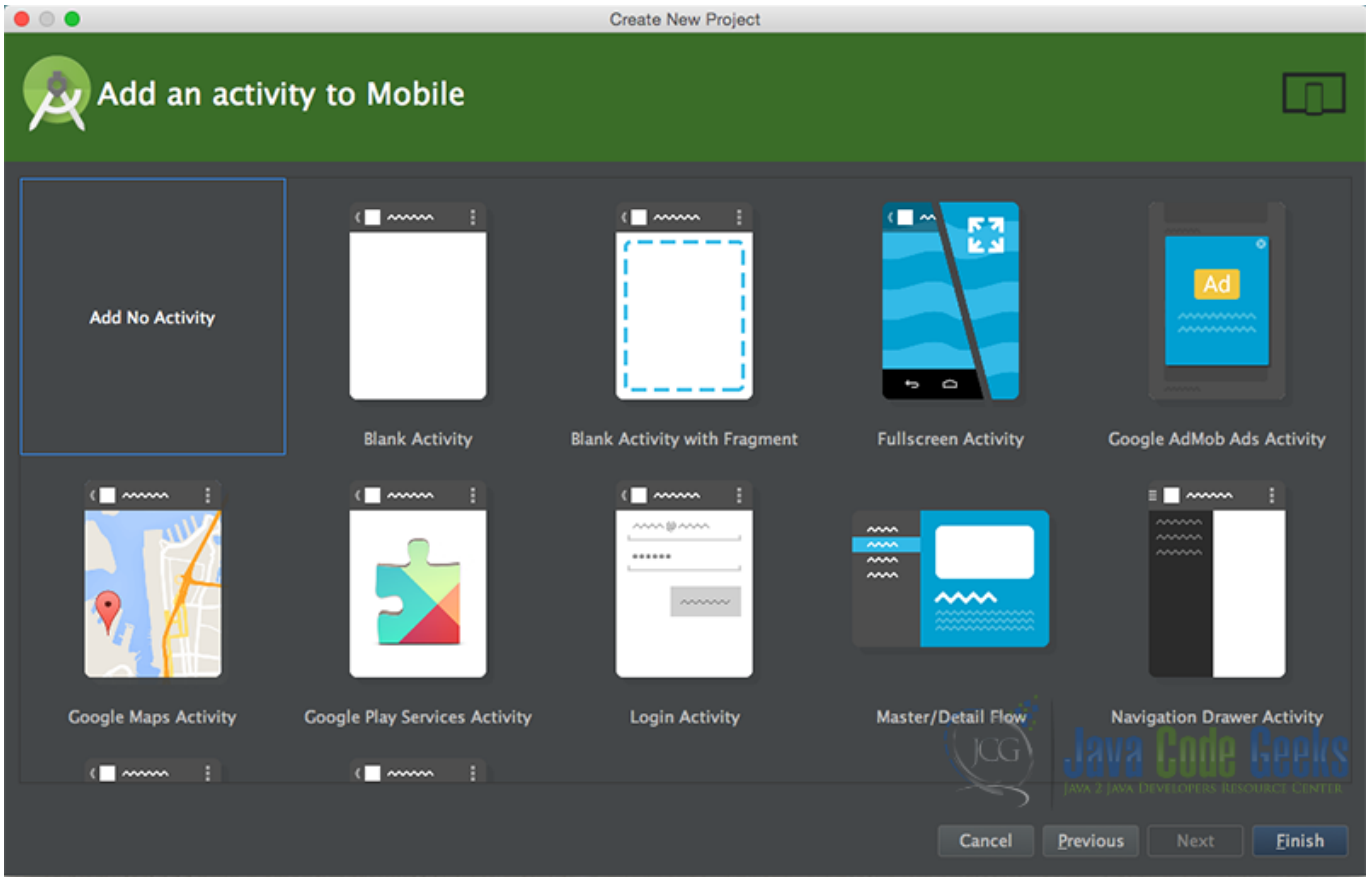

Figure 7.4: “Add an activity to Mobile”. Choose: “Add no activity”.

Now press finish, and our project has just been created!

7.2 Create the layout of the AndroidStackViewActivity

Add a new xml file inside `/res/layout` folder, with name `activity_main.xml`. We should have `/layout/activity_main.xml` file and paste the code below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffcc00">

    <StackView
        android:id="@+id/stack_view"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:animateLayoutChanges="true"></StackView>

</RelativeLayout>
```

7.3 Create the layout of the StackView items

Add a new xml file inside `/res/layout` folder, with name `item.xml`. We should have `/layout/item.xml` file and paste the code below.

item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#454545">

    <ImageView
        android:id="@+id/image"
        android:layout_width="420dp"
        android:layout_height="300dp"
        android:layout_centerHorizontal="true" />

</FrameLayout>
```

7.4 Create the source code of the StackItems

Add a new Java class inside `src/com.javacodegeeks.AndroidStackViewExample/` so that we are going to have the `src/com.javacodegeeks.AndroidStackViewExample/StackItems.java` file and paste the code below.

StackItems.java

```
package com.javacodegeeks.AndroidStackViewExample;

public class StackItems {
    Integer image;

    public StackItems(String name, Integer image) {
        this.image = image;
    }

    public int getImage() {
        return image;
    }

}
```

7.5 Create the source code of the StackAdapter

Add a new Java class inside `src/com.javacodegeeks.AndroidStackViewExample/` so that we are going to have the `src/com.javacodegeeks.AndroidStackViewExample/StackItems.java` file and paste the code below.

StackAdapter.java

```
package com.javacodegeeks.AndroidStackViewExample;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
```

```
import android.widget.ImageView;

import java.util.ArrayList;

public class StackAdapter extends BaseAdapter {

    ArrayList arrayList;
    LayoutInflater inflater;
    ViewHolder holder = null;

    public StackAdapter(Context context, ArrayList arrayList) {
        this.arrayList = arrayList;
        this.inflater = LayoutInflater.from(context);
    }

    @Override
    public int getCount() {
        return arrayList.size();
    }

    @Override
    public StackItems getItem(int pos) {
        return arrayList.get(pos);
    }

    @Override
    public long getItemId(int pos) {
        return pos;
    }

    @Override
    public View getView(int pos, View view, ViewGroup parent) {
        if (view == null) {
            view = inflater.inflate(R.layout.item, parent, false);
            holder = new ViewHolder();
            holder.image = (ImageView) view.findViewById(R.id.image);
            view.setTag(holder);
        } else {
            holder = (ViewHolder) view.getTag();
        }
        holder.image.setBackgroundResource(arrayList.get(pos).getImage());

        return view;
    }

    public class ViewHolder {
        ImageView image;
    }

}
```

7.6 Create the source code of the AndroidStackViewActivity

Add a new Java class inside `src/com.javacodegeeks.AndroidStackViewExample/` so that we are going to have the `src/com.javacodegeeks.AndroidStackViewExample/AndroidStackViewActivity.java` file and paste the code below.

AndroidStackViewActivity.java

```
package com.javacodegeeks.AndroidStackViewActivity;

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

import java.util.ArrayList;

public class AndroidStackViewActivity extends Activity {

    private static StackView stackView;
    private static ArrayList list;

    private static final Integer[] icons = {R.drawable.jellybean, R.drawable.kitkat,
            R.drawable.lollipop, R.drawable.marshmellow, R.drawable.nougat};

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

        stackView = (StackView) findViewById(R.id.stack_view);
        list = new ArrayList();

        for (int i = 0; i < icons.length; i++) {
            list.add(new StackItems("Item " + i, icons[i]));
        }
        StackAdapter adapter = new StackAdapter(AndroidStackViewActivity.this, list);
        stackView.setAdapter(adapter);
        adapter.notifyDataSetChanged();
    }
}
```

7.7 AndroidManifest.xml

The AndroidManifest.xml of our project has no special permissions.

AndroidManifest.xml

```
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.AndroidStackViewExample">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".AndroidStackViewActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest
>
```

7.8 build.gradle

The build.gradle of our project is simple.

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.javacodegeeks.AndroidStackViewExample"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
}
```

7.9 Build, compile and run

When we build, compile and run our project, the main AndroidStackViewActivity should look like this:

Figure 7.5: This is how our application looks.

7.10 Download the Android Studio Project

This was an example of Android StackView.

Download You can download the full source code of this example here: **AndroidStackViewExample**

Chapter 8

Android ViewPager Example

One of the most popular views in Android is the ViewPager, a view that helps us slide from one "screen" to another like a "slideshow". Android ViewPager view can be found in many cases, in tutorial or introductory screens, on tab views and galleries.

We are going to create an AppCompatActivity, and we are going to add a ViewPager reference and in it, as well as a FragmentPagerAdapter that will help us navigate between the Fragments that our ViewPager will consist of and that we are going to create. Then, we are going to make instances of Fragments that will be added in our Adapter. The Android ViewPager has default swipe-gestures from one "screen" to another, and we do not need to create any gestures in order to transit between the pages.

So, in this example, we are going to show, how we can implement an Android ViewPager. Let's start.

For our example will use the following tools in a Windows 64-bit or an OS X platform:

- JDK 1.7
- Android Studio 1.3.2
- Android SDK 6.0

Let's take a closer look:

8.1 Create a New Android Studio Project

Open Android Studio and choose Start a new Android Studio Project in the welcome screen.

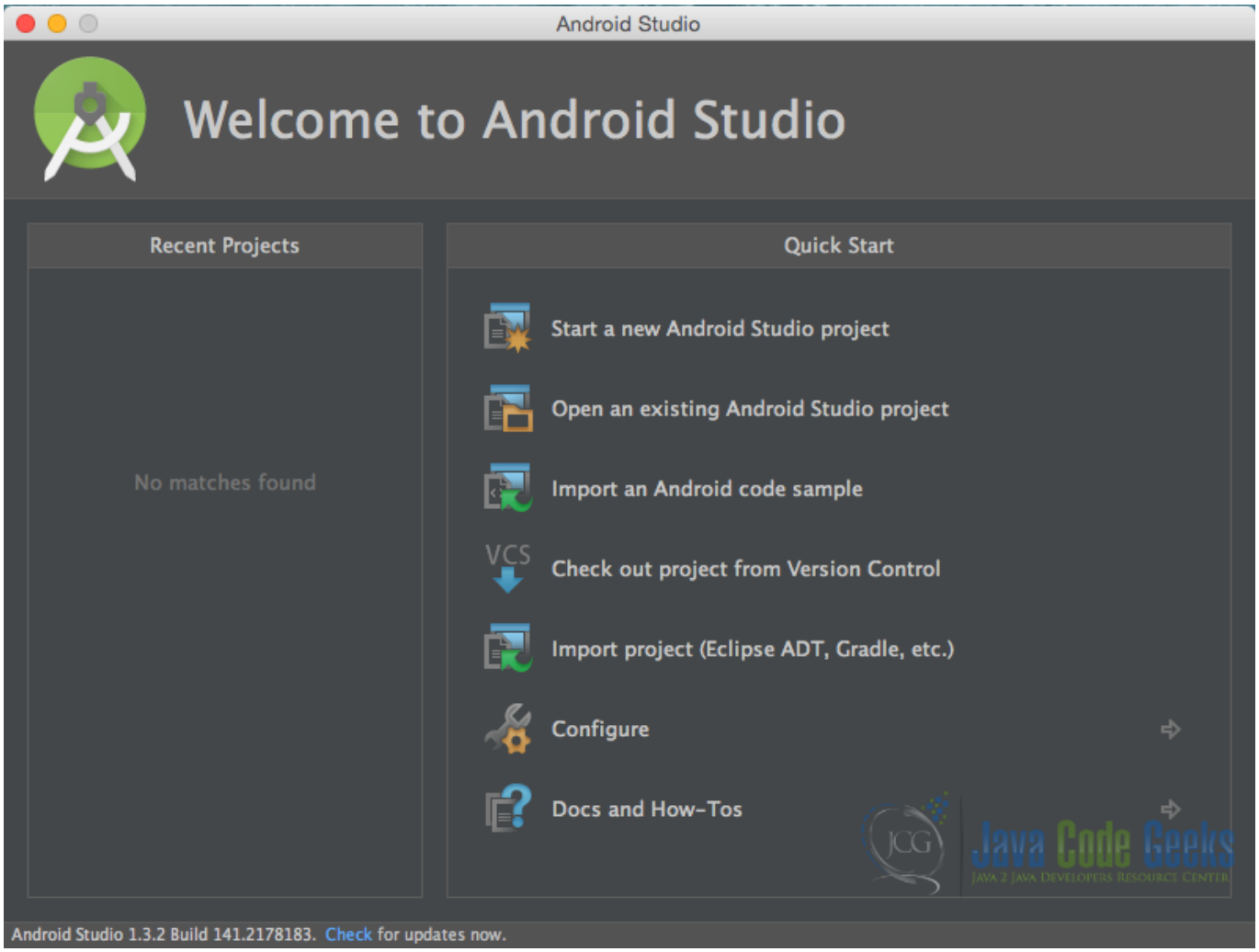

Figure 8.1: Welcome to Android Studio screen. Choose Start a new Android Studio Project.

Specify the name of the application, the project and the package.

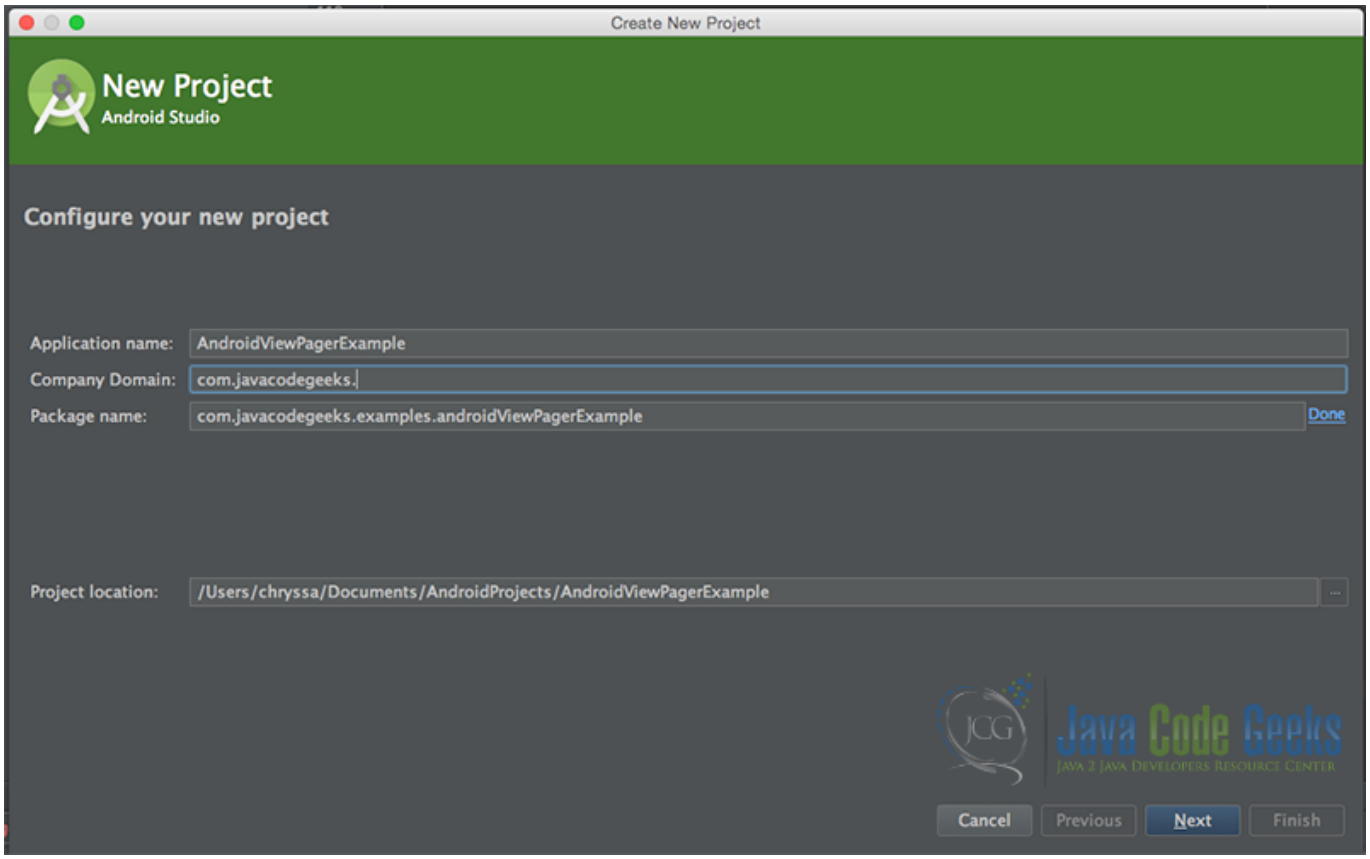

Figure 8.2: Configure your new project screen. Add your application name and the projects package name.

In the next window, select the form factors your app will run on.

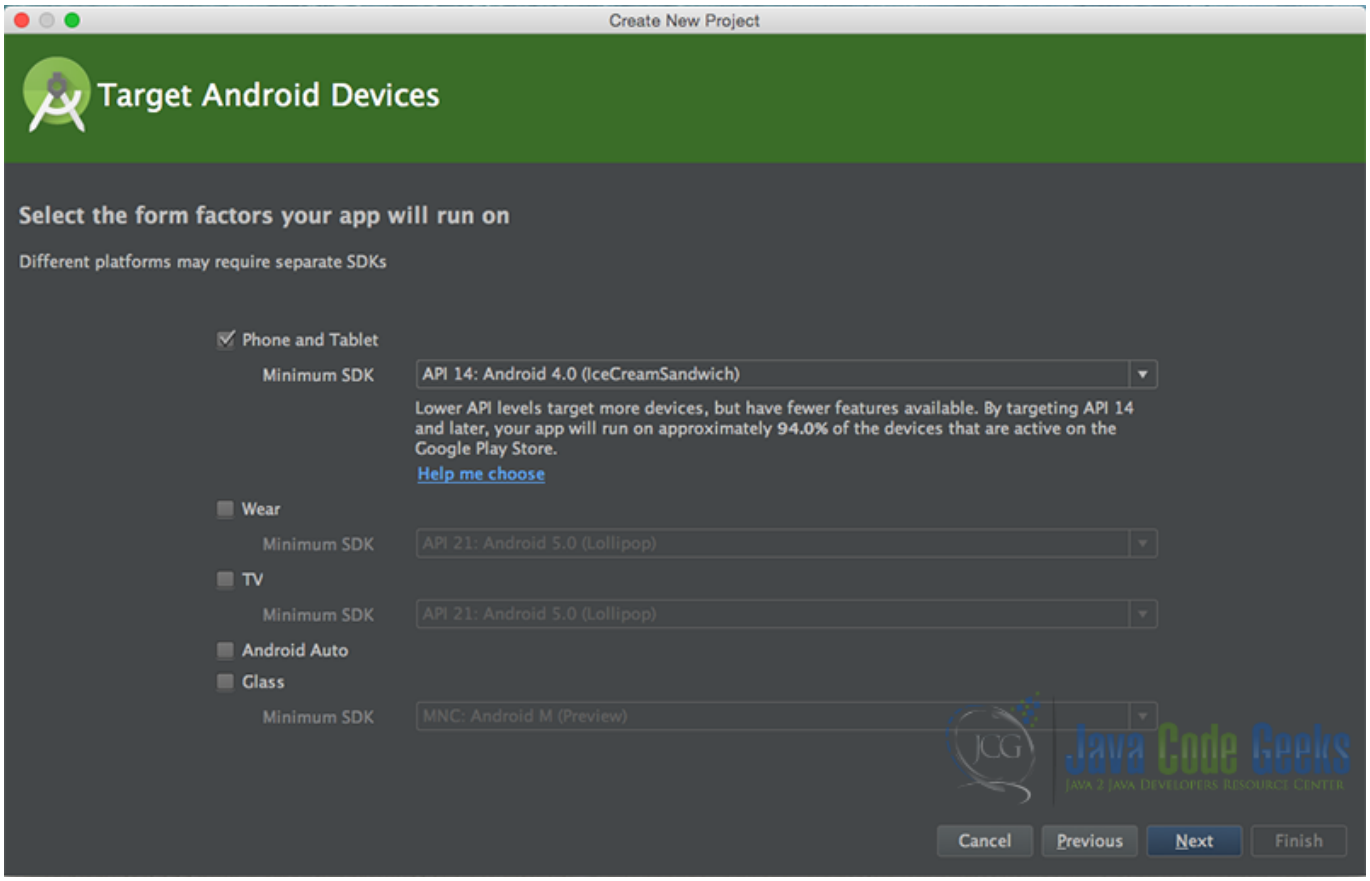

Figure 8.3: Target Android Devices screen.

In the next window you should choose Add no activity. In this example, we are going to create our Activity.

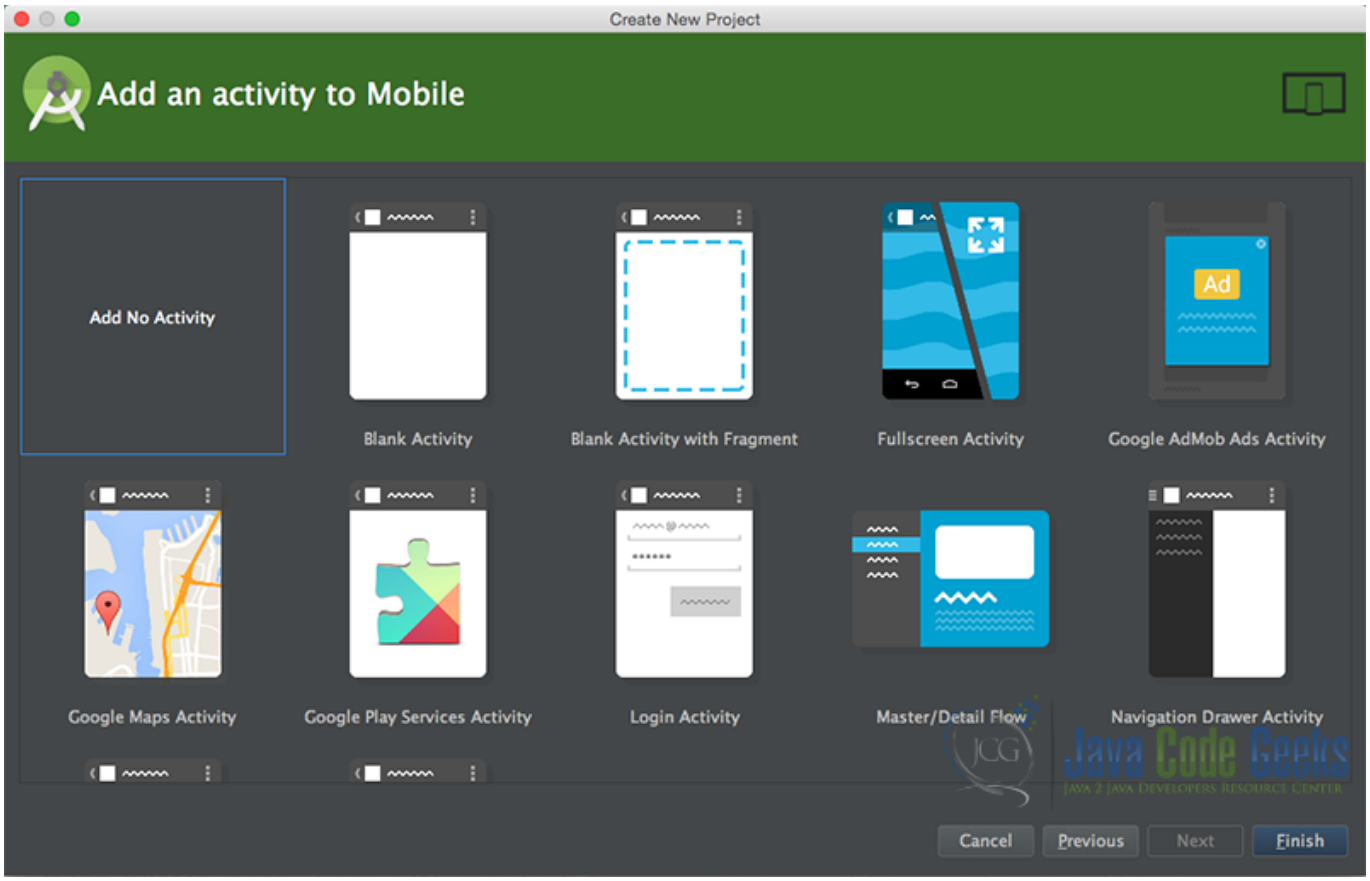

Figure 8.4: Add an activity to Mobile. Choose: Add no activity.

Now, our project has just been created!

8.2 Create the layout of the main AndroidViewPagerExample

The AndroidViewPagerExample is the main Activity of our example, and this is the Activity in which we are going to create a ViewPager instance and set its FragmentPagerAdapter. Now, we are going to make the layout of this Activity, which will be a simple layout xml for the AndroidViewPagerExample.class, that consists of a LinearLayout with vertical orientation, that includes a ViewPager.

Add a new xml file inside `/res/layout` folder, with name activity_main.xml. We should have the `/res/layout/activity_main.xml` file and paste the code below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    xmlns:custom="https://schemas.android.com/apk/res-auto"
    xmlns:tools="https://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:background="@drawable/background_blue"
    android:orientation="vertical">

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        tools:context=".MainActivity"/>

</LinearLayout>
```

8.3 Create the source code of the main AndroidViewPagerExample Activity

Add a new Java class inside `src/com.javacodegeeks.androidViewPagerExample/` so that we are going to have the `src/com.javacodegeeks.androidViewPagerExample/AndroidViewPagerExample.java` file and paste the code below.

AndroidViewPagerExample.java

```
package com.javacodegeeks.examples.androidViewPagerExample;

import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.os.Bundle;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;

public class AndroidViewPagerExample extends AppCompatActivity {

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

        ViewPager pager = (ViewPager) findViewById(R.id.pager);
        pager.setAdapter(new MyPagerAdapter(getSupportFragmentManager()));
    }

    private class MyPagerAdapter extends FragmentPagerAdapter {

        public MyPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public android.support.v4.app.Fragment getItem(int pos) {
            switch (pos) {
                case 0:
                    return FragmentViewPager.newInstance(getString(R.string.title_section1) ↩
                        , R.drawable.rock);
                case 1:
                    return FragmentViewPager.newInstance(getString(R.string.title_section2) ↩
                        , R.drawable.paper);
                case 2:
                    return FragmentViewPager.newInstance(getString(R.string.title_section3) ↩
                        , R.drawable.scissors);
                default:
                    return FragmentViewPager.newInstance(getString(R.string.title_section1) ↩
                        , R.drawable.rock);
            }
        }

        @Override
        public int getCount() {
```

```
            return 3;
        }
    }
}
```

8.4 Create the layout of the main FragmentViewPager

The FragmentViewPager is the Fragment of our example, from which we are going to recreate instances and add them in the ViewPager. Now, we are going to make the layout of this Fragment, which will be a simple layout xml for the FragmentViewPager.class, that consists of a LinearLayout with vertical orientation, that includes a TextView and an ImageView.

Add a new xml file inside `/res/layout` folder, with name fragment_main.xml. We should have the `/res/layout/fragment_main.xml` file and paste the code below.

fragment_main.xml

```
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    xmlns:tools="https://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#00000000"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="44dp"
        android:textColor="#ffffff"
        android:textSize="60dp" />

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="44dp" />

</LinearLayout>
```

8.5 Create the source code of the main FragmentViewPager support.v4.app.Fragment

Add a new Java class inside `src/com.javacodegeeks.androidViewPagerExample/` so that we are going to have the `src/com.javacodegeeks.androidViewPagerExample/FragmentViewPager.java` file and paste the code below.

FragmentViewPager.java

```
package com.javacodegeeks.examples.androidViewPagerExample;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

public class FragmentViewPager extends android.support.v4.app.Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle ↩
        savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_main, container, false);

        TextView tv = (TextView) v.findViewById(R.id.title);
        tv.setText(getArguments().getString("text"));

        ImageView imageView = (ImageView) v.findViewById(R.id.image);
        imageView.setBackgroundResource(getArguments().getInt("img"));

        return v;
    }

    public static FragmentViewPager newInstance(String text, int image) {

        FragmentViewPager f = new FragmentViewPager();
        Bundle b = new Bundle();
        b.putString("text", text);
        b.putInt("img", image);

        f.setArguments(b);

        return f;
    }
}
```

8.6 Android Manifest

The AndroidManifest.xml of our project is simple and contains the main Activity of our example:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
    package="com.javacodegeeks.examples.androidViewPagerExample" >

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

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

</manifest
```

```
>
```

8.7 Composing build.gradle file

We should add the AppCompat V7 support support library in our project. We can this as a dependency to our application via build.gradle file.

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.javacodegeeks.examples.androidViewPagerExample"
        minSdkVersion 14
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules. ↩
                pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.1.1'
}
```

8.8 Build, compile and run

When we build, compile and run our project, the main AndroidViewPagerExample should look like this:

Figure 8.5: This is the first Fragment of our ViewPager.

Figure 8.6: This is the second Fragment of our ViewPager.

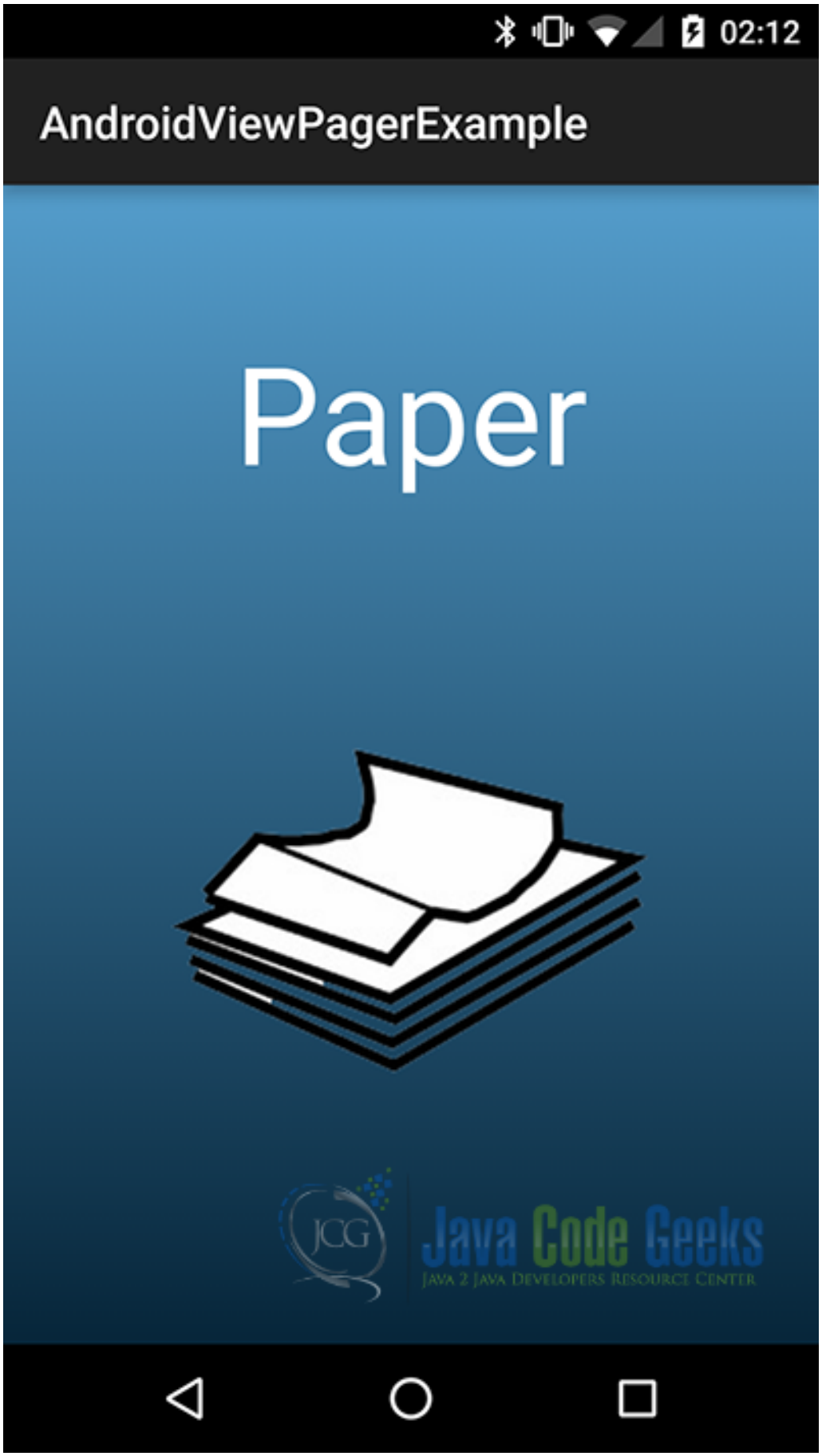

Figure 8.7: This is the third Fragment of our ViewPager.

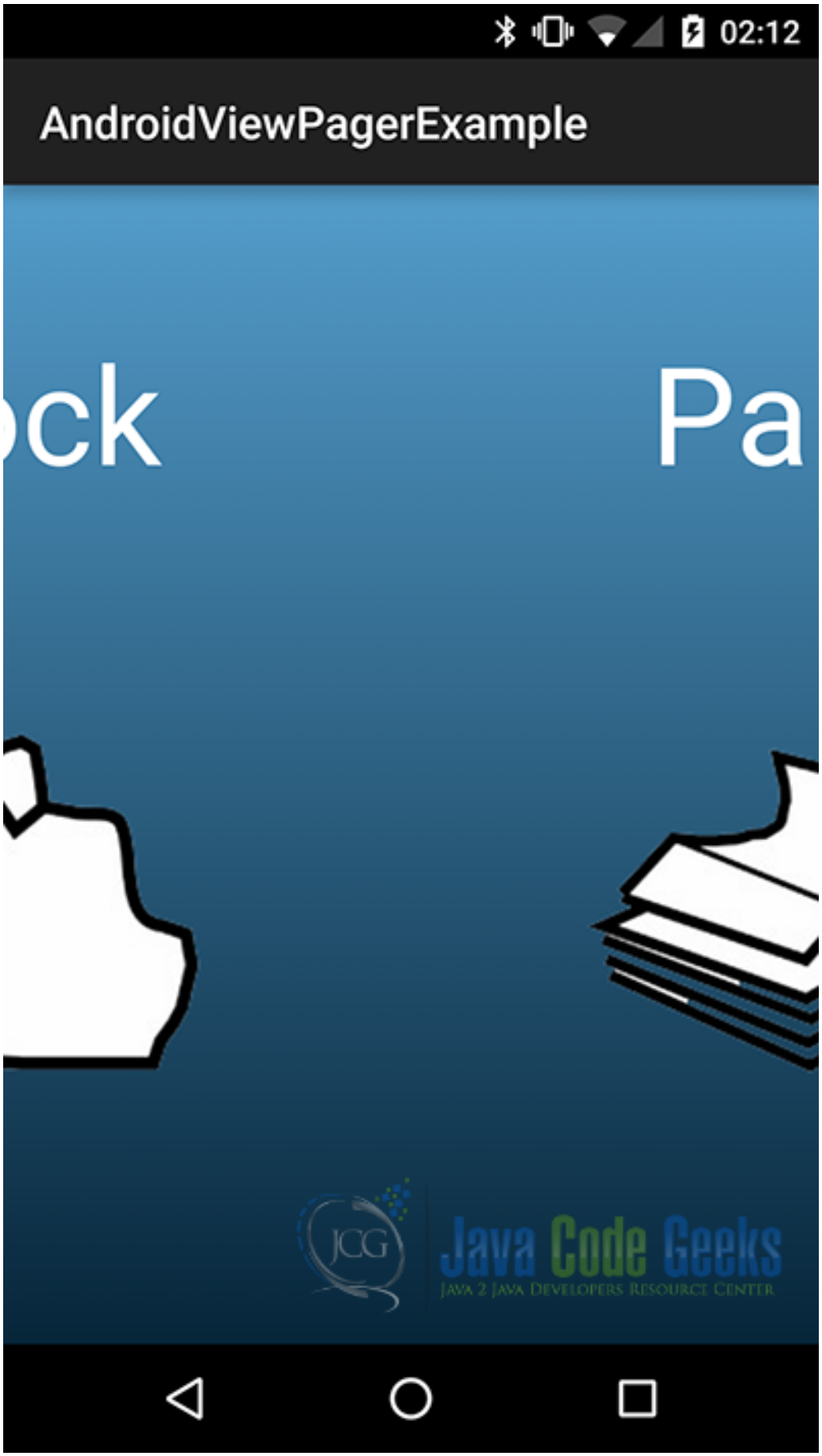

Figure 8.8: This is the transition between two Fragments.

8.9 Download the Android Studio Project

This was an example of Android ViewPager Example.

Download You can download the full source code of this example here: **AndroidViewPagerExample**