raywenderlich.com

Kotlin For Android: An Introduction

Eunice Obugyei on October 19, 2016

When creating native apps for Android, you will normally use the Java programming language for your logic. Whilst Java is a battle hardened language, it isn't without its issues.

When you run a Java application, the app is compiled into a set of instructions called Bytecode and runs in a virtual machine. Over the past several years, a number of new programming languages have been introduced to also run on the Java virtual machine. Whilst the resulting app looks the same for the virtual machine, the idea is the language features can help developers write simpler code and fix some of Javas issues.

JetBrains, known for IntelliJ IDEA (Android Studio is based on IntelliJ), has introduced the Kotlin language.

Kotlin is a statically-typed programming language that runs on the JVM. It can also be compiled to JavaScript source code. Kotlin has some amazingly cool features!

In this Kotlin for Android tutorial, you'll learn:

- How to setup your Kotlin environment.
- How to work with both Java and Kotlin in the same project.
- What makes Kotlin so exciting as a new language.

Note: This tutorial assumes you're experienced in Android development with Java. If you're new to the Android world, have big questions about the starter project or are not familiar with Android Studio, please have a look at our Android tutorials.

Why Kotlin For Android?

Since Android took the world by storm, developers have had no alternatives to Java app development. Although its usage is widespread, Java comes with a lot of historical baggage.

Java 8 solved some language issues and corrected even more with Java 10. In order to get much of a benefit from the corrections made in these two releases, you have to set the minimum SDK to Android 24 just to use Java 8, which isn't an option for many developers. For almost everybody, Java 10 isn't even on the radar.

Kotlin aims to fill that gap of a missing modern language for the Android platform. There are a few core tenets that Kotlin lives by; it strives to be:

1. Concise to reduce the amount of boilerplate code you need to write.
2. Expressive to make your code more readable and understandable.
3. Safe to avoid entire classes of errors such as null pointer exceptions.
4. Versatile for building server-side applications, Android apps or frontend code running in the browser.
5. Interoperable to leverage existing frameworks and libraries of the JVM with 100 percent Java interoperability.

Above all, it's a new language! What could be more exciting? iOS developers can't have all the fun. :]

Getting Started

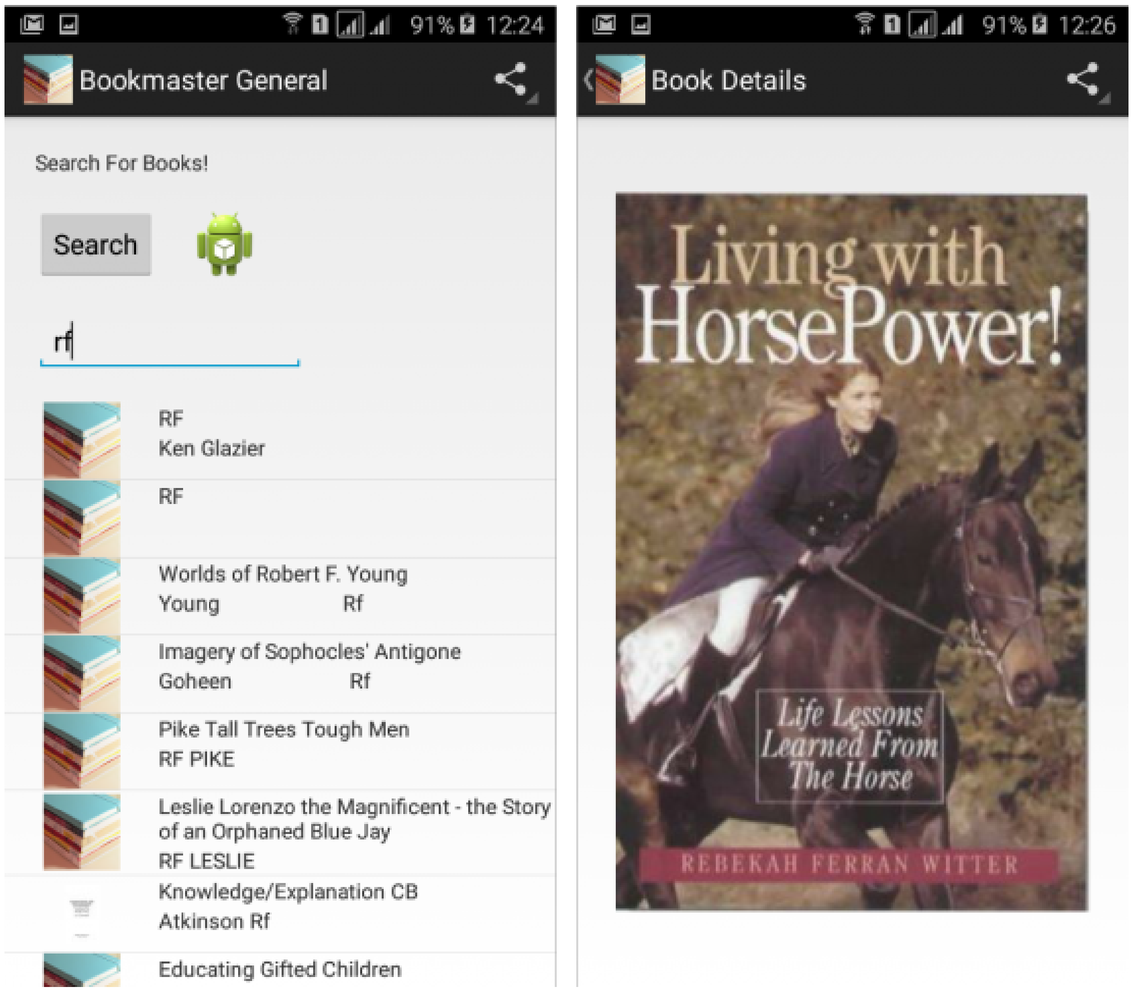
Download the starter project. Extract and open the starter project in Android Studio.

You'll be working with this simple app that allows users to search for books, see book covers, and share books with friends to explore Kotlin.

It contains three source code files; take a moment to get familiar with them:

- **MainActivity.java:** an `Activity` that displays the screen for searching and displaying a list of books.
- **DetailActivity.java:** an `Activity` that displays the book cover for the ID passed to it.
- **JSONAdapter.java:** a custom `BaseAdapter` that transforms a JSON object into a list view item.

Build and run the project to see what you're working with.

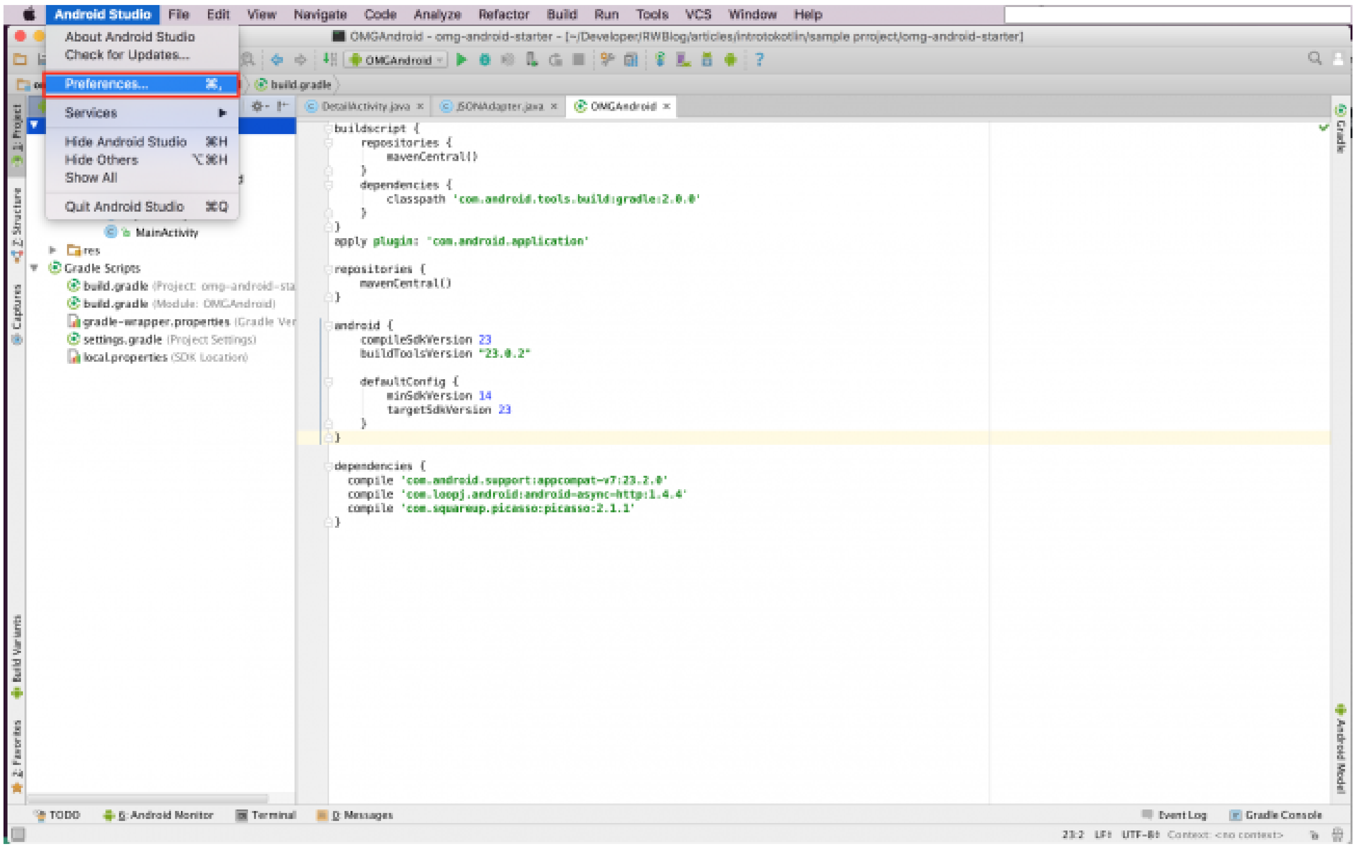

91% 12:24
Bookmaster General
Search For Books!
Search
rf
RF
Ken Glazier
RF
Worlds of Robert F. Young
Young Rf
Imagery of Sophocles' Antigone
Goheen Rf
Pike Tall Trees Tough Men
RF PIKE
Leslie Lorenzo the Magnificent - the Story of an Orphaned Blue Jay
RF LESLIE
Knowledge/Explanation CB
Atkinson Rf
Educating Gifted Children
91% 12:26
Book Details
Living with HorsePower!
Life Lessons Learned From The Horse
REBEKAH FERRAN WITTER

Setting up Your Environment

By default, Android Studio has no idea what to do with Kotlin, so the first step is to install the Kotlin plugin and configure Kotlin in your project.

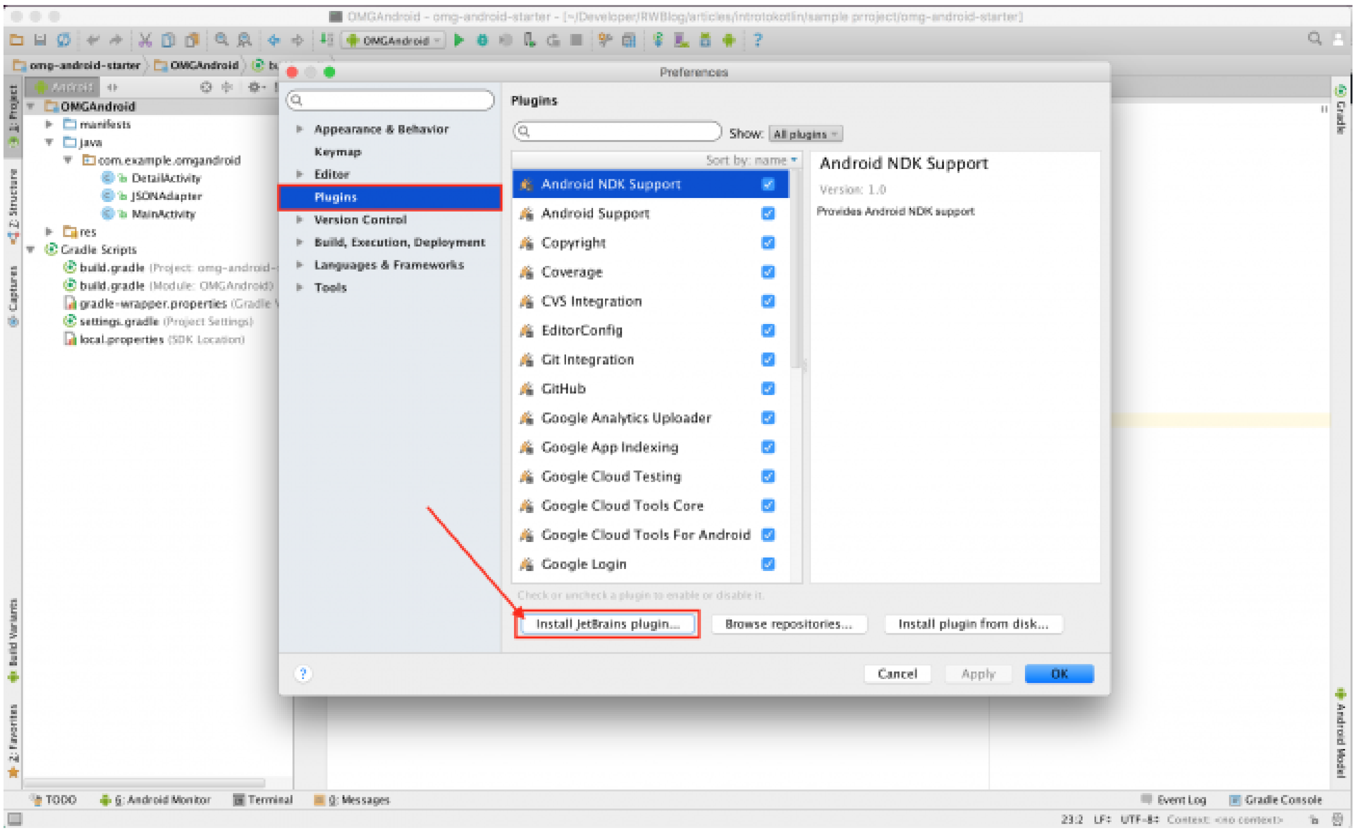
Installing the Kotlin Plugin

Go to **Android Studio\Preferences** and select the **Plugins** entry.

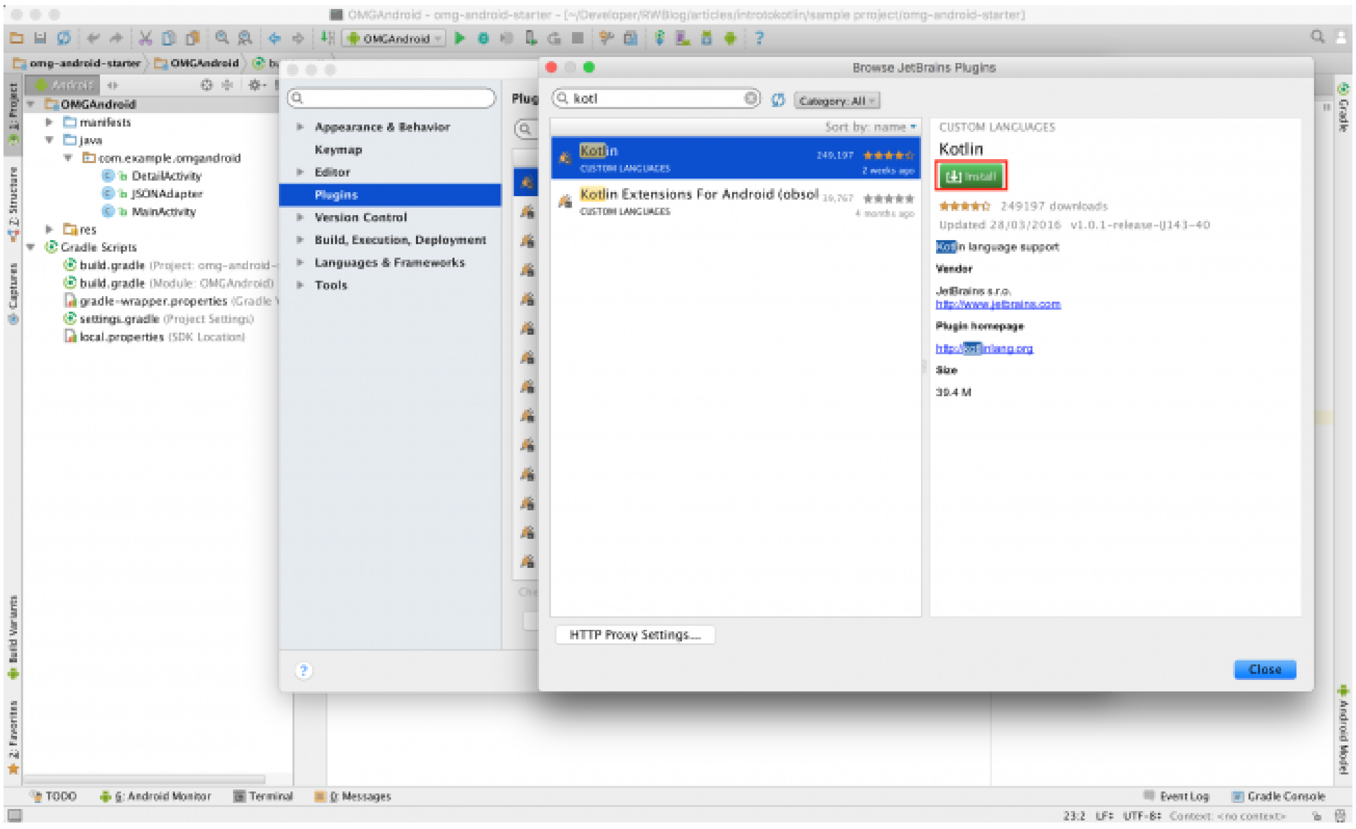


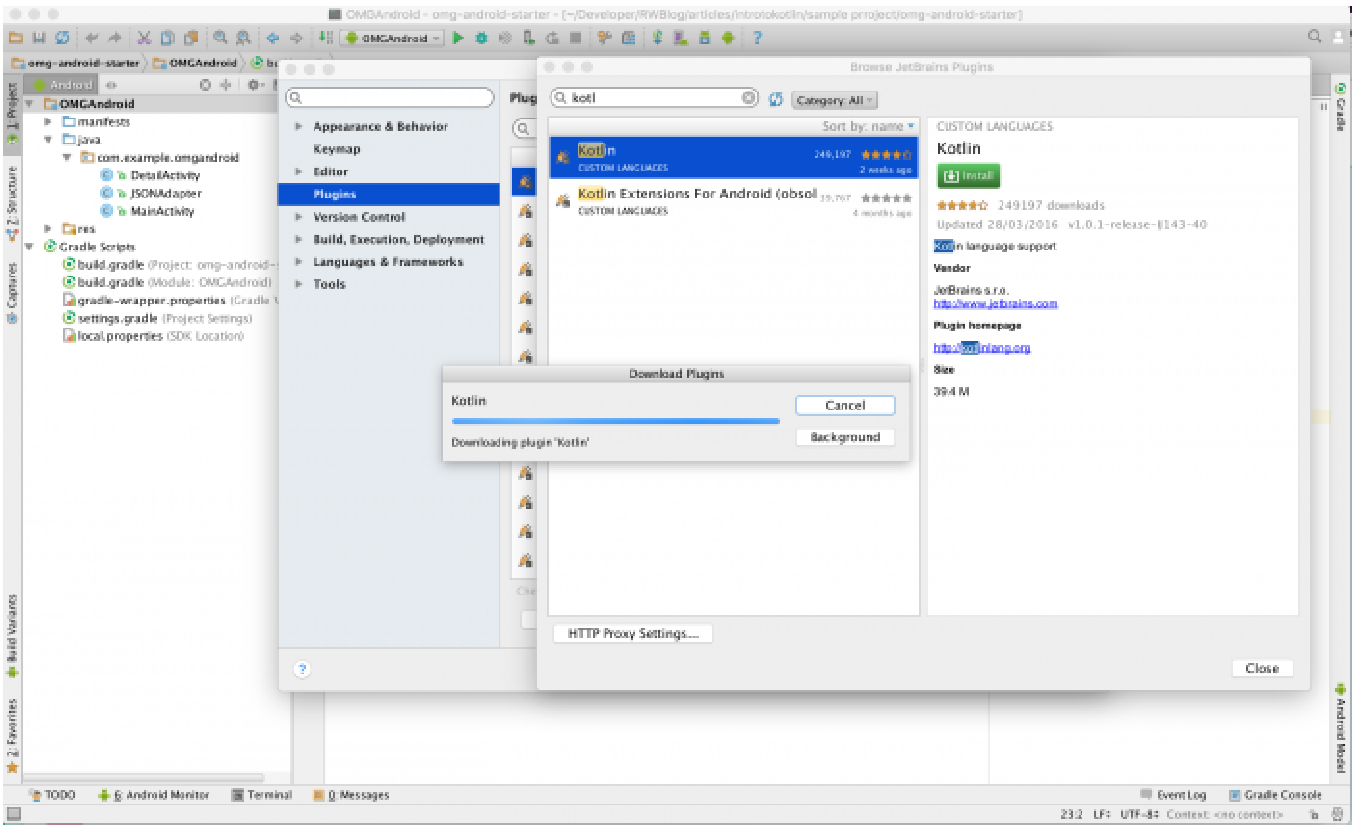
```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.0.0'
    }
}
apply plugin: 'com.android.application'

repositories {
    mavenCentral()
}

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 23
    }
}

dependencies {
  compile 'com.android.support:appcompat-v7:23.2.0'
  compile 'com.loopj.android:android-async-http:1.4.4'
  compile 'com.squareup.picasso:picasso:2.1.1'
}
```

On the **Plugins** screen, click on **Install JetBrains plugin…**

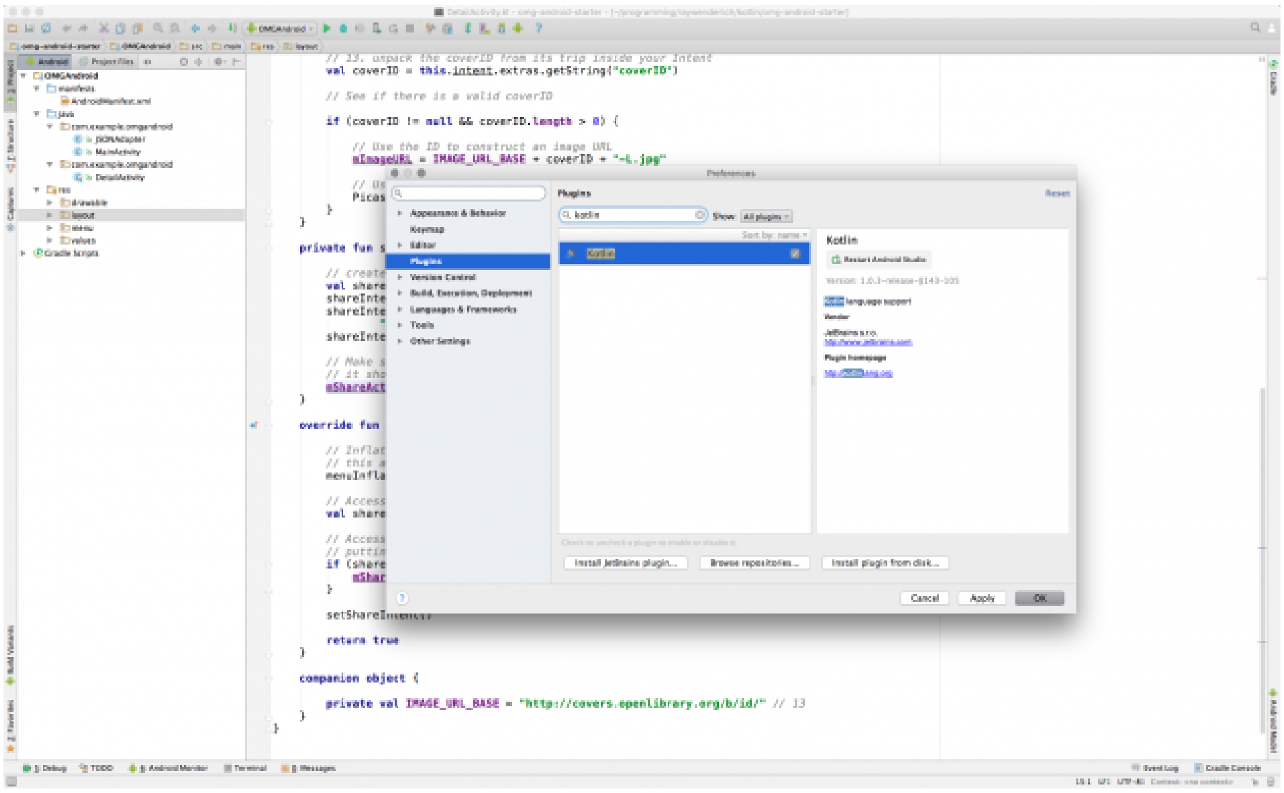


Search for and select **Kotlin** from the list and click **Install**.





When you’re finished with downloading and installing, the next step is following the prompts to restart the IDE.

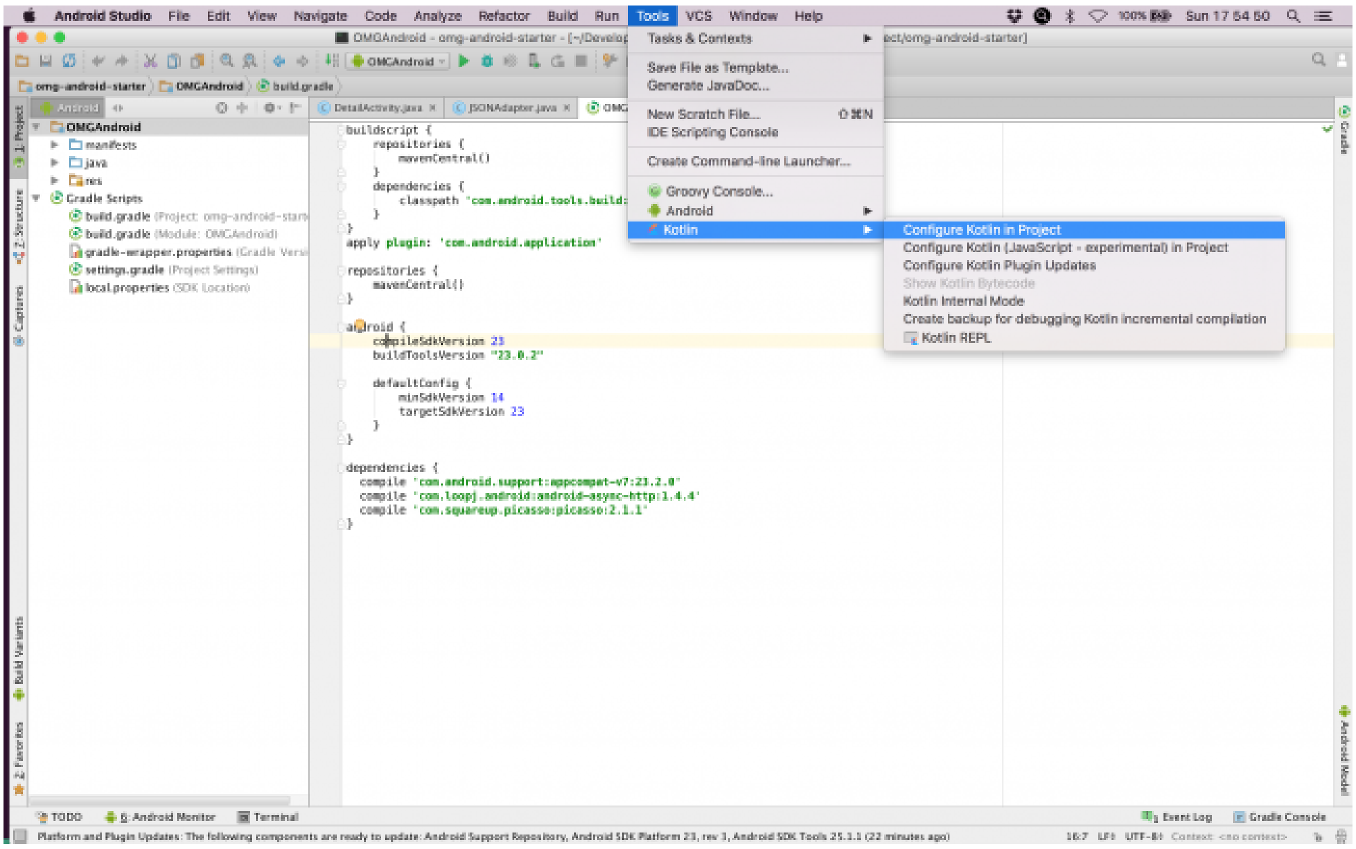


```
        // 13. unpack the coverID from its trip inside your Intent
        val coverID = this.intent.extras.getString("coverID")

        // See if there is a valid coverID

        if (coverID != null && coverID.length > 0) {

            // Use the ID to construct an image URL
            mImageURL = IMAGE_URL_BASE + coverID + "-L.jpg"

            // Us
            Picas
        }
    }

    private fun s

        // create
        val share
        shareInte
        shareInte

        shareInte

        // Make s
        // it sho
        mShareAct
    }

    override fun

        // Inflat
        // this a
        menuInfla

        // Access
        val share

        // Access
        // puttin
        if (share
            mShar
        }

        setShareIntent(

        return true
    }

    companion object {

        private val IMAGE_URL_BASE = "http://covers.openlibrary.org/b/id/" // 13
    }
}
```

Configure Kotlin in Project

Now the IDE knows what to do with Kotlin, but your project app doesn't, so your next move is to modify the project's build configuration.

Go to **Tools\Kotlin\Configure Kotlin in Project**.

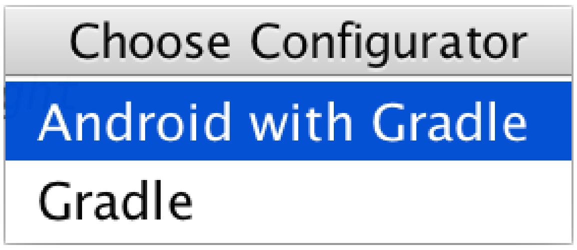


```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:
    }
}
apply plugin: 'com.android.application'

repositories {
    mavenCentral()
}

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 23
    }
}

dependencies {
  compile 'com.android.support:appcompat-v7:23.2.0'
  compile 'com.loopj.android:android-async-http:1.4.4'
  compile 'com.squareup.picasso:picasso:2.1.1'
}
```

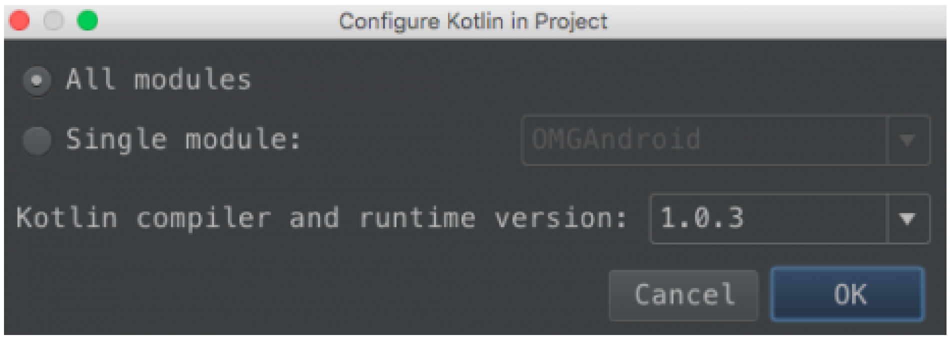
Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Tasks & Contexts
Save File as Template...
Generate JavaDoc...
New Scratch File... ⇧⌘N
IDE Scripting Console
Create Command-line Launcher...
Groovy Console...
Android
Kotlin
Configure Kotlin in Project
Configure Kotlin (JavaScript - experimental) in Project
Configure Kotlin Plugin Updates
Show Kotlin Bytecode
Kotlin Internal Mode
Create backup for debugging Kotlin incremental compilation
Kotlin REPL
OMGAndroid
manifests
java
res
Gradle Scripts
build.gradle (Project: omg-android-starter)
build.gradle (Module: OMGAndroid)
gradle-wrapper.properties (Gradle Version)
settings.gradle (Project Settings)
local.properties (SDK Location)

Platform and Plugin Updates: The following components are ready to update: Android Support Repository, Android SDK Platform 23, rev 3, Android SDK Tools 25.1.1 (22 minutes ago)

Select **Android with Gradle** from the **Choose Configurator** popup that appears.



On the **Configure Kotlin in Project** popup, select the plugin version you want to use (at the time of writing this tutorial, the current version is 1.0.3) and click **OK**.



Kotlin compiler and runtime version: 1.0.3

This action will make a number of changes to your **build.gradle** files.

build.gradle (Project: omg-android-starter):

```
buildscript {
  ext.kotlin_version = '1.0.3' // 1
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'com.android.tools.build:gradle:2.1.3'
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version" // 2

    // NOTE: Do not place your application dependencies here; they belong
    // in the individual module build.gradle files
  }
}

allprojects {
  repositories {
    jcenter()
  }
}
```

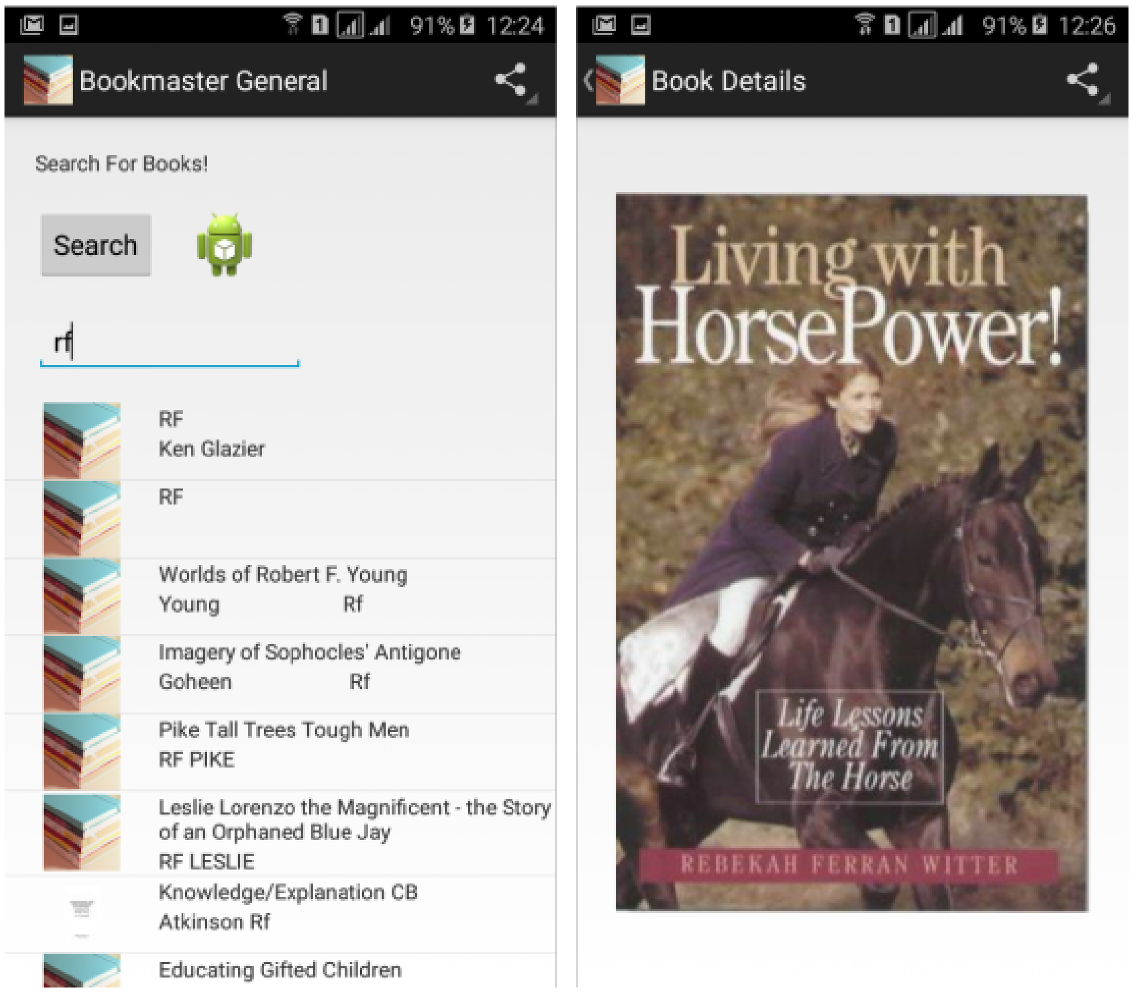
build.gradle (Module: OMG Android):

```

apply plugin: 'com.android.application'
apply plugin: 'kotlin-android' // 3

android {
    compileSdkVersion 23
    buildToolsVersion "24.0.2"

    defaultConfig {
        minSdkVersion 14
        targetSdkVersion 23
    }
  sourceSets {
    main.java.srcDirs += 'src/main/kotlin' // 4
  }
}

dependencies {
  compile 'com.android.support:appcompat-v7:23.2.0'
  compile 'com.loopj.android:android-async-http:1.4.4'
  compile 'com.squareup.picasso:picasso:2.1.1'
  compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version" // 5
}
repositories {
  mavenCentral()
}
```

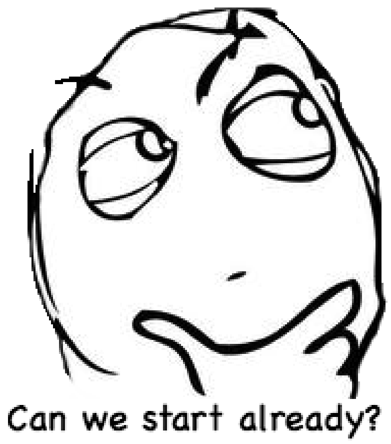
Let's go over this step-by-step:

1. Declares the version of Kotlin configured in the project
2. Declares a classpath dependency artifact that contains the Kotlin Gradle plugin with the version declared earlier
3. Specifies the use of the Kotlin Android plugin via `apply plugin command`
4. Defines that source files found in **src/main/kotlin** will be compiled. Strictly speaking, gradle will compile source Kotlin files found in **src/main/java**, but it's nice to put to put Kotlin files in a Kotlin directory.
5. Added the Kotlin Standard Library as a compile time dependency to the project

Click on **Sync Now** to build the project. Build and run.


91% 12:24
Bookmaster General
Search For Books!
Search
rf
RF
Ken Glazier
RF
Worlds of Robert F. Young
Young Rf
Imagery of Sophocles' Antigone
Goheen Rf
Pike Tall Trees Tough Men
RF PIKE
Leslie Lorenzo the Magnificent - the Story of an Orphaned Blue Jay
RF LESLIE
Knowledge/Explanation CB
Atkinson Rf
Educating Gifted Children
91% 12:26
Book Details
Living with HorsePower!
Life Lessons Learned From The Horse
REBEKAH FERRAN WITTER

Nothing changed visually but you've put all the plumbing in place to support Kotlin in your Android project.

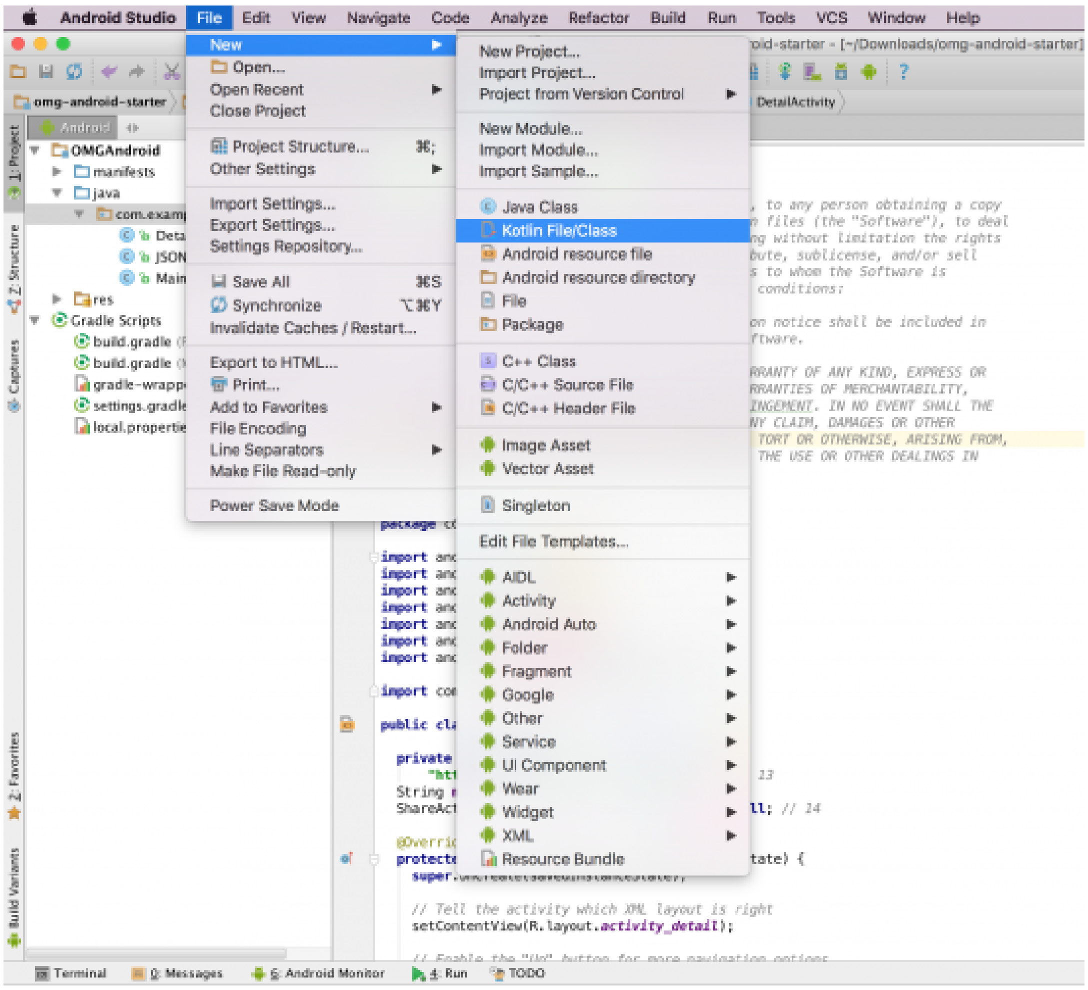


Working with Java and Kotlin in the Same Project

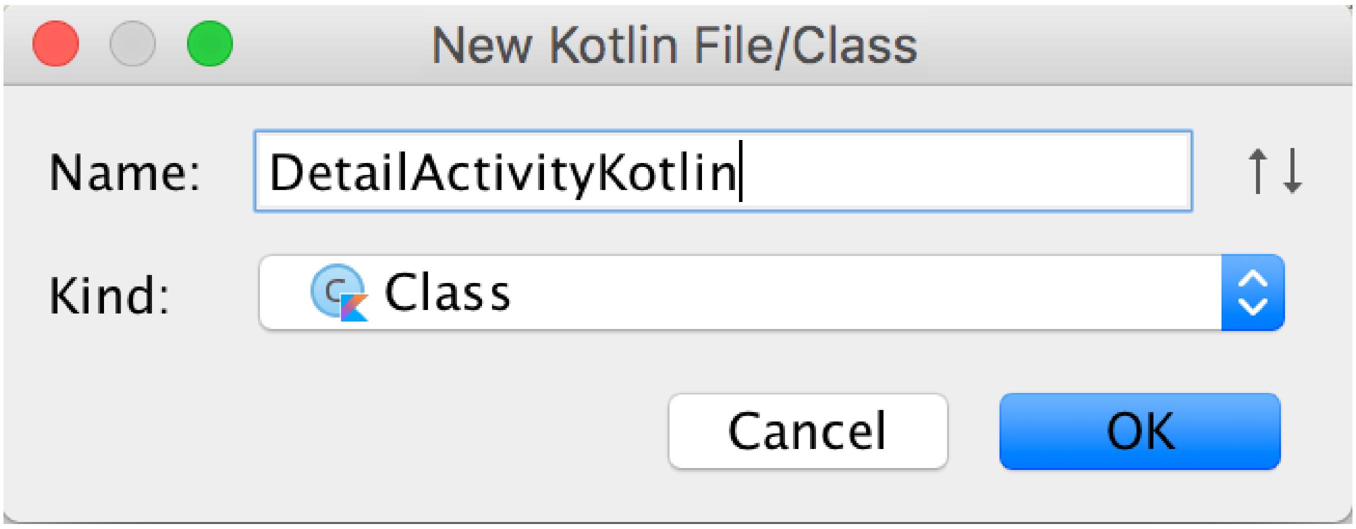
One of the most amazing qualities of Kotlin is how it can coexist with Java on a project. Java code can be called from Kotlin and vice versa.

From this point of the tutorial forward, you'll be translating the **DetailActivity** class in Kotlin.

Single click the `com.example.omgandroid` package in the Project panel on the left-hand side. With the package selected, go to **File\New\Kotlin File/Class** to create a new Kotlin class. (Without the package selected, you won't see the Kotlin file option).



On the **New Kotlin File/Class** popup, select **Class** in the **Kind** field and enter **DetailActivityKotlin** as the class name. Click **OK**.



Your new class should look like this:

```
    package com.example.omgandroid

    class DetailActivityKotlin {
    }
```

A few things to note here:

1. As you may have noticed in the above code, classes in Kotlin are declared using the keyword **`class`** — just like in Java.
2. The default visibility modifier in Kotlin is **`public`**.
3. Classes and methods are final by default. You can declare them **open** if you want extensibility.

Since Kotlin is Java interoperable, you can use existing Java frameworks and libraries in your Kotlin code files.

First place the following import statements at the top of the file:

```
  import android.app.Activity
  import android.os.Bundle
```

Then make the class a subclass of **Activity**.

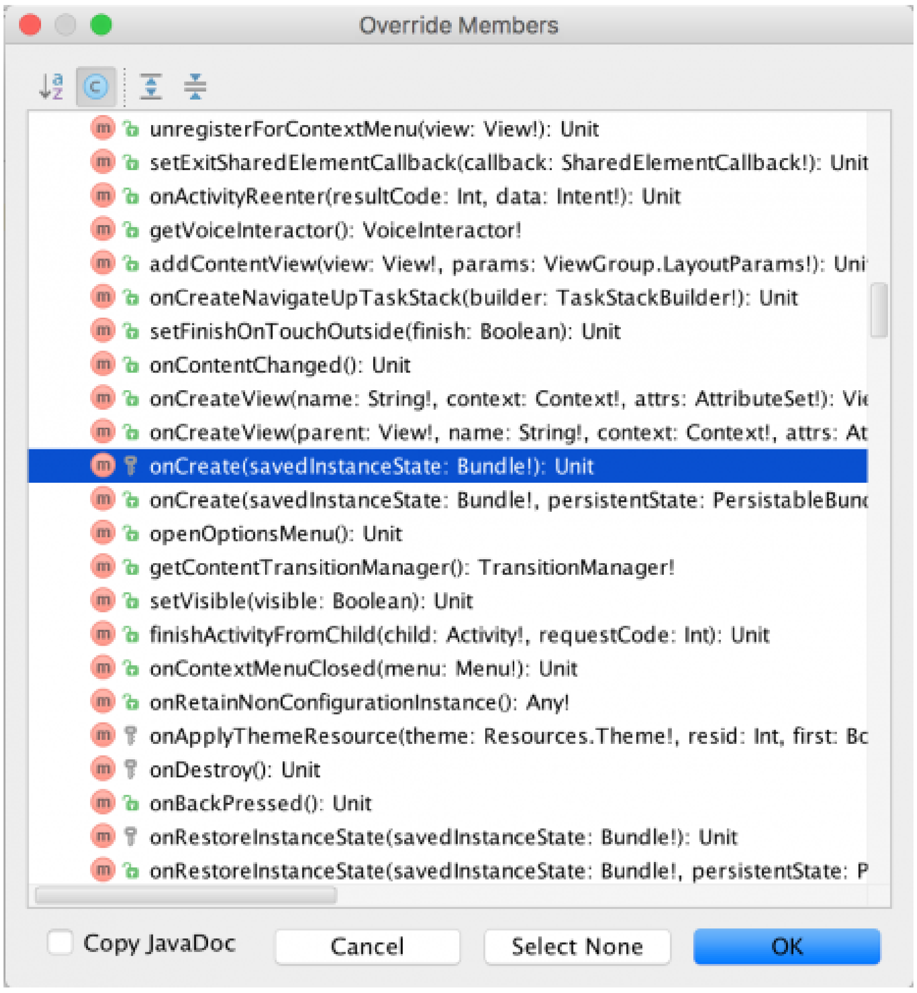
```
class Main2Activity : Activity() {

}
```

Note that you do this in Kotlin a little differently from how you do it in Java. In Kotlin, you append **:NameOfParentClass()** to the subclass declaration.

Now override **Activity**'s **onCreate()** method. It will look something like this.

```
import android.app.Activity
import android.os.Bundle

class DetailActivityKotlin: Activity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
  }
}
```

Note: You can use Android Studio's code generation functionality to generate the **onCreate** method signature with **control + O**. Press **control + O** to see a popup with all overridable methods for the class you're in.



```
unregisterForContextMenu(view: View!): Unit
setExitSharedElementCallback(callback: SharedElementCallback!): Unit
onActivityReenter(resultCode: Int, data: Intent!): Unit
getVoiceInteractor(): VoiceInteractor!
addContentView(view: View!, params: ViewGroup.LayoutParams!): Uni
onCreateNavigateUpTaskStack(builder: TaskStackBuilder!): Unit
setFinishOnTouchOutside(finish: Boolean): Unit
onContentChanged(): Unit
onCreateView(name: String!, context: Context!, attrs: AttributeSet!): Vie
onCreateView(parent: View!, name: String!, context: Context!, attrs: At
onCreate(savedInstanceState: Bundle!): Unit
onCreate(savedInstanceState: Bundle!, persistentState: PersistableBunc
openOptionsMenu(): Unit
getContentTransitionManager(): TransitionManager!
setVisible(visible: Boolean): Unit
finishActivityFromChild(child: Activity!, requestCode: Int): Unit
onContextMenuClosed(menu: Menu!): Unit
onRetainNonConfigurationInstance(): Any!
onApplyThemeResource(theme: Resources.Theme!, resid: Int, first: Bc
onDestroy(): Unit
onBackPressed(): Unit
onRestoreInstanceState(savedInstanceState: Bundle!): Unit
onRestoreInstanceState(savedInstanceState: Bundle!, persistentState: P
```

Open **MainActivity.java** and replace the **DetailActivity** reference in **onItemClick()** with **DetailActivityKotlin**.

Your intent creation line should change from:

```
Intent detailIntent = new Intent(this, DetailActivity.class);
```

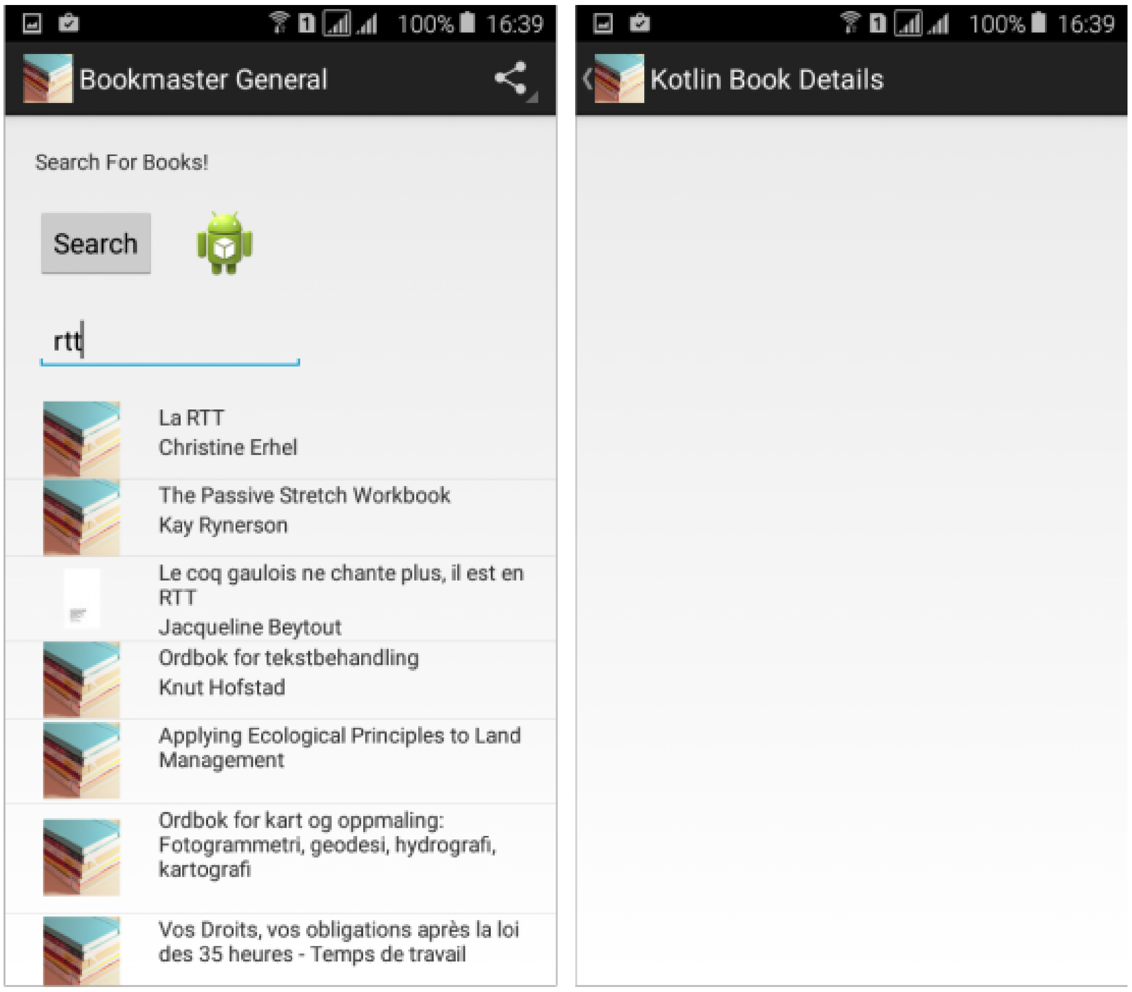
To this:

```
Intent detailIntent = new Intent(this, DetailActivityKotlin.class);
```

Just like you would do for a Java Activity, you need to declare your Kotlin Activity in **AndroidManifest.xml**. Add the following code under the **DetailActivity** declaration:

```
    <activity
         android:name=".DetailActivityKotlin"
         android:label="@string/activity_details_kotlin"
         android:parentActivityName=".MainActivity">
       <meta-data
           android:name="android.support.PARENT_ACTIVITY"
           android:value=".MainActivity"/>
     </activity>
```

Build and run. Select a book from the list so you can see that empty screen with the title *Kotlin Book Details*.



rtt

La RTT
Christine Erhel

The Passive Stretch Workbook
Kay Rynerson

Le coq gaulois ne chante plus, il est en RTT
Jacqueline Beytout

Ordbok for tekstbehandling
Knut Hofstad

Applying Ecological Principles to Land Management

Ordbok for kart og oppmaling: Fotogrammetri, geodesi, hydrografi, kartografi

Vos Droits, vos obligations après la loi des 35 heures - Temps de travail

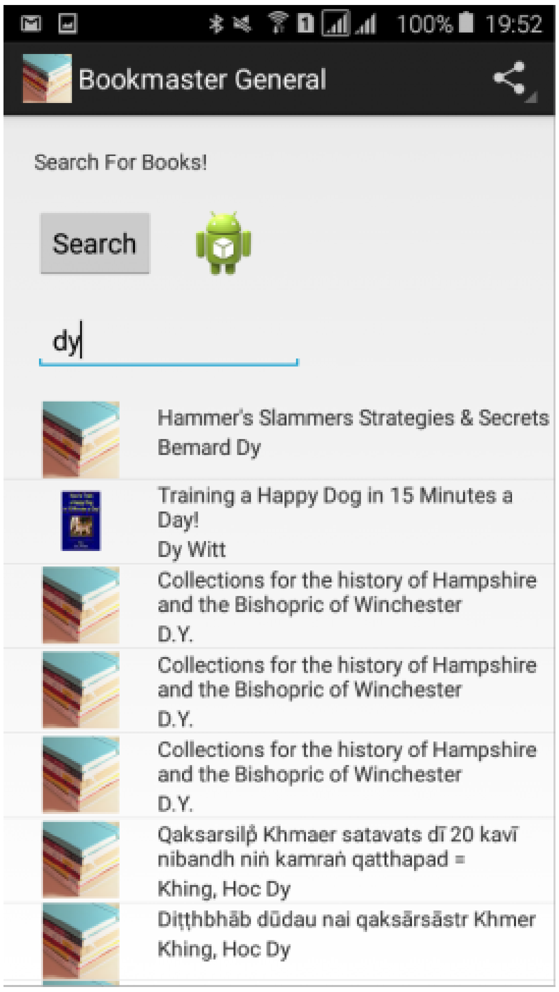
How Cool is Kotlin?

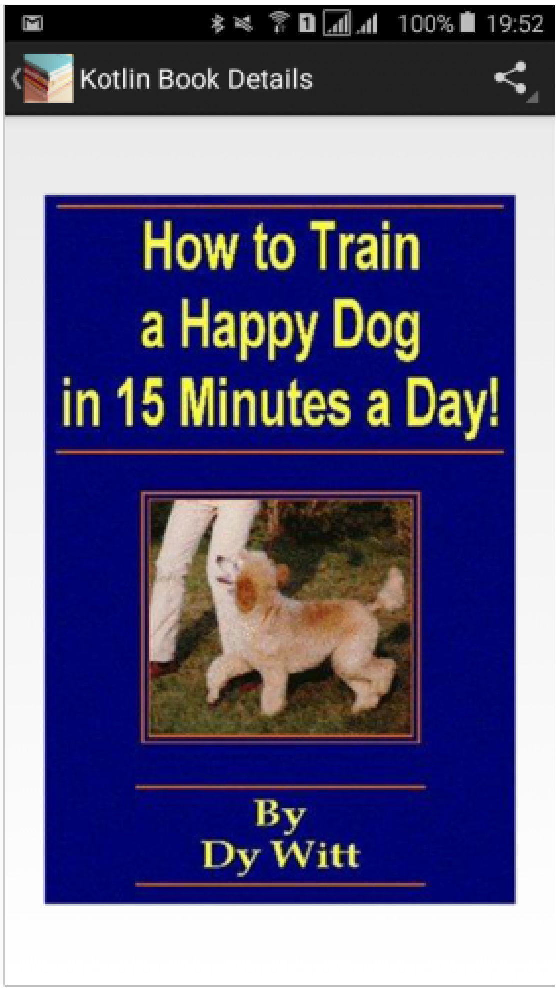
Before you dive deeper into Kotlin's features, go back to **DetailActivityKotlin.kt** and replace the contents of the file with the following:

```
package com.example.omgandroid

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.view.Menu
import android.widget.ImageView
import android.widget.ShareActionProvider
import com.squareup.picasso.Picasso

class DetailActivityKotlin: Activity() {

  private val IMAGE_URL_BASE = "http://covers.openlibrary.org/b/id/"
  internal var mImageURL = ""
  internal var mShareActionProvider: ShareActionProvider? = null

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_detail)

    actionBar?.setDisplayHomeAsUpEnabled(true)

    val imageView = findViewById(R.id.img_cover) as ImageView

    val coverId = this.intent.extras.getString("coverID")

    val len = coverId?.length ?: 0
    if (len > 0) {
      mImageURL = IMAGE_URL_BASE + coverId + "-L.jpg"

Picasso.with(this).load(mImageURL).placeholder(R.drawable.img_books_loading).into(imageView)
    }
  }

  private fun setShareIntent() {

    val shareIntent = Intent(Intent.ACTION_SEND)
    shareIntent.type = "text/plain"
    shareIntent.putExtra(Intent.EXTRA_SUBJECT, "Book Recommendation!")
    shareIntent.putExtra(Intent.EXTRA_TEXT, mImageURL)

    mShareActionProvider?.setShareIntent(shareIntent)
  }

  override fun onCreateOptionsMenu(menu: Menu): Boolean {

    menuInflater.inflate(R.menu.main, menu)

    val shareItem = menu.findItem(R.id.menu_item_share)

    mShareActionProvider = shareItem!!.actionProvider as ShareActionProvider

    setShareIntent()

    return true
  }
}
```

On the surface, the code resembles Java, but there are some Kotlin language specifics that you'll get into in the next section.

Build and run, select a book and see if you get a cover this time. Oh, look, you do!


100% 19:52
Bookmaster General
Search For Books!
Search
dy
Hammer's Slammers Strategies & Secrets
Bemard Dy
Training a Happy Dog in 15 Minutes a Day!
Dy Witt
Collections for the history of Hampshire and the Bishopric of Winchester
D.Y.
Collections for the history of Hampshire and the Bishopric of Winchester
D.Y.
Collections for the history of Hampshire and the Bishopric of Winchester
D.Y.
Qaksarsilp̊ Khmaer satavats dī 20 kavī nibandh niṅ kamraṅ qatthapad =
Khing, Hoc Dy
Diṭṭhbhāb dūdau nai qaksārsāstr Khmer
Khing, Hoc Dy



Null Safety

One of the leading points of frustration with most programming languages, including Java, is accessing a member of a null reference. A null reference occurs when you declare an object variable but haven't given it a value. When the program runs and tries to access that variable it doesn't know where to look for it memory because it doesn't exist.

The most common result of this is your application come to an abrupt halt and crashes! You might be familiar with Java's "almighty" **NullPointerException**. Apologies in advance for any flashbacks! :]

One of Kotlin's greatest features is that its type system aims to eliminate the **NullPointerException** (a goal known as void safety).

In Kotlin, the only possible causes of a **NullPointerException** are:

- External Java code did it
- An explicit call to throw NullPointerException()
- Usage of the **!!** operator (which will be explained shortly)
- Some data inconsistency in regards to initialization

Nullable Types and Non-Null Types

Kotlin has **nullable** and **non-null** types. If you don't declare a variable as nullable, then you cannot assign it a null value. This is enforced by the compiler meaning it's much harder to unintentionally crash your app.

In contrast to Java, all variables must be initialized at the point of declaration.

To declare a variable as nullable, you have to append a **?** to its type at the point of declaration as you see in this **mShareActionProvider** declaration:

```
internal var mShareActionProvider: ShareActionProvider? = null
```

Safe Calls

To access a property or method on a nullable variable in Java, you would first do a null check. You can see this in `DetailActivity.java`:

```
if (mShareActionProvider != null) {
  mShareActionProvider.setShareIntent(shareIntent)
}
```

With Kotlin, you can simplify the above expression with the use of a safe call operator (**?.**). The property or method is only called when the nullable variable is *not null*.

```
mShareActionProvider?.setShareIntent(shareIntent)
```

Here, `setShareIntent` is only called when the `mShareActionProvider` property is not null.

The !! Operator

As stated earlier, this is one of possible causes of the dreaded **NullPointerException**. If you're absolutely sure the object is not null, feel free to use the (**!!**) operator to dereference your object.

You can see an example of this in `setShareIntent()`:

```
mShareActionProvider = shareItem!!.actionProvider as ShareActionProvider
```

In here, the **actionProvider** is retrieved if the **shareItem** variable is not null, but a **NullPointerException** is thrown when the **shareItem** variable is null.

The Elvis Operator

The Elvis Operator (**?:**) looks like the ternary `if` operator in Java but works differently. You can see an example of this when trying to get the length of the cover ID:

```
val len = coverId?.length ?: 0
```

If the expression to the left of the Elvis operator is not null, the results of the expression are returned. Otherwise, the it returns the expression to the right.

Just like an `if-else` statement, Elvis only evaluates the expression on the right if the one on the left side is null.

Type Inference

Kotlin also supports type inference, meaning the compiler can assume its type from the initializer when a variable is declared and initialized. For example, the types of the `IMAGE_URL_BASE` and `mImageURL` variables are inferred from their initializers.

```
private val IMAGE_URL_BASE = "http://covers.openlibrary.org/b/id/"
internal var mImageURL = ""
```

The compiler tracks the inferred type of each variable (each is a `String`), and any subsequent values assigned to the variable must also be of that type (`String`).

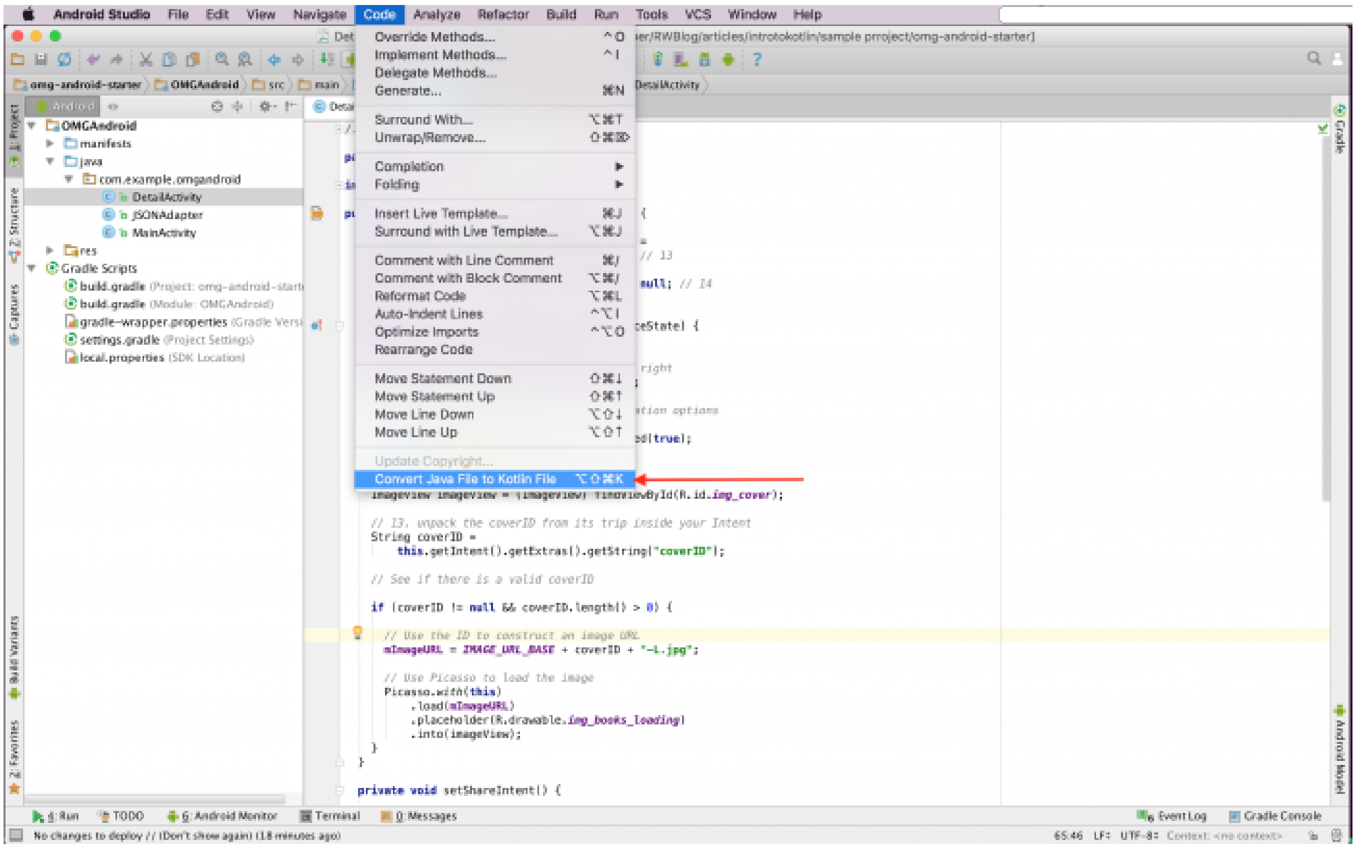
The Coolest of Them All

Already thinking of rewriting your Java project in Kotlin? Don't stress — the Kotlin plugin has you covered.

Since Kotlin is a programming language made by developers for developers, it's designed to make your life as easy as possible. The Kotlin plugin even has a handy tool that allows you to convert a Java source file to Kotlin.

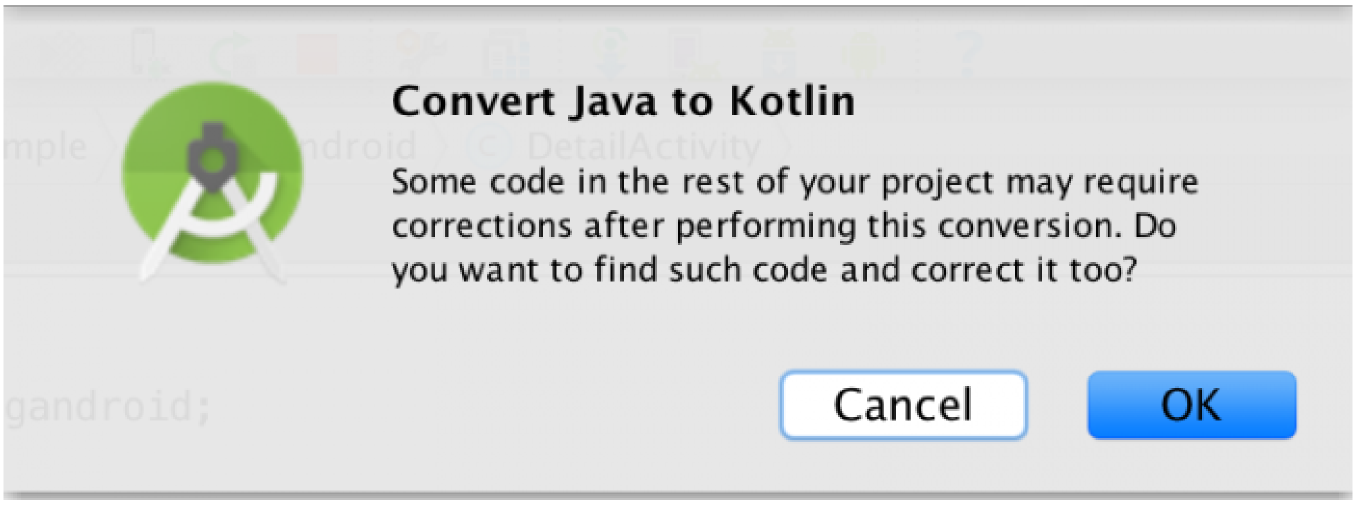
Take this sanity-saving feature for a test drive by converting the **DetailActivity.java** file to Kolin.

Open the **DetailActivity.java** class and go to **Code\Convert Java File to Kotlin File**.



```
ImageView imageView = (ImageView) findViewById(R.id.img_cover);

// 13. unpack the coverID from its trip inside your Intent
String coverID =
    this.getIntent().getExtras().getString("coverID");

// See if there is a valid coverID

if (coverID != null && coverID.length() > 0) {

  // Use the ID to construct an image URL
  mImageURL = IMAGE_URL_BASE + coverID + "-L.jpg";

  // Use Picasso to load the image
  Picasso.with(this)
      .load(mImageURL)
      .placeholder(R.drawable.img_books_loading)
      .into(imageView);
 }
}

private void setShareIntent() {
```

Click **OK** on the **Convert Java to Kotlin** screen. This will replace the Java file with a Kotlin one!



Convert Java to Kotlin

Some code in the rest of your project may require corrections after performing this conversion. Do you want to find such code and correct it too?

That's it. You've converted a Java class into a Kotlin class. :]

Where To Go From Here?

Congratulations! You just learned about the Kotlin programming language and some of it's amazing features, re-coded a Java Activity in Kotlin, and used the Kotlin plugin to convert a Java source file into a Kotlin source file.

Download the final project for this tutorial here.

I suggest reading up further on Null Safety in Kotlin in the documentation.

You can also use the Kotlin online compiler to try out code samples and improve your knowledge of the language.

You've only scratched the surface of the amazing possibilities with Kotlin. If you're excited by what you've read here, you can checkout topics such as Data Classes, Extensions, Lambdas, or String Templates if you need to satisfy your appetite for knowledge.

I hope you enjoyed this Kotlin for Android tutorial, and if you have any questions or comments, please join the forum discussion below!

Team

Each tutorial at www.raywenderlich.com is created by a team of dedicated developers so that it meets our high quality standards. The team members who worked on this tutorial are:

A
E

E
V

Team Lead
Matt Luedke

Eunice Obugyei

Eunice is a software engineer at DreamOval, currently focusing on Mobile development.