Appendix

Appendix: Utilities

Contents:

This appendix is a collection of tasks you might need to do as you develop the apps in the Android Developer Fundamentals practical codelabs. The tasks are not specific to one practical.

How to copy and rename a project

For some of the practical codelabs, you need to make a copy of a project before you make new changes to the project. You might also want to copy a project to use some of its code in a new project. In either case, you can copy the existing project, then rename and refactor the new project's components to use the new project's name.

In the instructions below, substitute your actual project names for **ExistingProject** and **NewProject**.

1. Copy the project

1. On your computer's file system (not in Android Studio), make a copy of the **ExistingProject** directory.
2. Rename the copied directory to **NewProject**.

2. Rename and refactor the project components

The old name of the project, **ExistingProject**, still appears throughout the packages and files in the **NewProject** project.

Here's how to change the file and the package references in your app to the new name:

1. Start Android Studio.
2. Select **Open an existing Android Studio project**. If you already have a project open in Android Studio, select **File > Open**.
3. Navigate to the **NewProject** directory, select it, and click **OK**.
4. Select **Build > Clean Project** to remove the auto-generated files.
5. Click the **1:Project** side-tab and choose **Android** from the drop-down menu to see your files in the Project: Android view.
6. Expand the **app > java** folder.
7. Right-click **com.example.android.existingproject** and select **Refactor > Rename**. A **Warning** dialog is displayed. Select **Rename package.**
8. The **Rename** dialog is displayed. Give a new name to your project.
9. Select **Search in comments and strings** and **Search for text occurrences**. Click **Refactor**.
10. The **Find: Refactoring Preview** pane appears, showing the code to be refactored. Click **Do Refactor**.
11. Expand the **res > values** folder and double-click the **strings.xml** file.
12. Change the `app_name` string value to `"New Project"`.

3. Update the build.gradle and AndroidManifest.xml files

Each app you create must have a unique application ID, as defined in the app's `build.gradle` file. The steps above should have changed the `build.gradle` file. To make sure, check the file and sync the project with the Gradle file by following these steps:

1. In Android Studio **Project** pane, expand **Gradle Scripts** and open **build.gradle (Module: app)**.
2. Under `defaultConfig`, check to make sure that the value of the `applicationID` key is **"com.example.android.newproject"**. If the value isn't correct, change it manually.
3. Click **Sync Now** in the top-right corner of the Android Studio window.

Tip: Another way to sync your Gradle files is to select **Tools > Android > Sync Project with Gradle Files**.

To make sure that the app name and package name are correct in the `AndroidManifest.xml` file, follow these steps:

1. Expand the **app > manifests** folder and double-click **AndroidManifest.xml**.
2. Check that the `package` name is correct. It should be `"com.example.android.newproject"`. If not, change it manually.
3. Find the statement below. If necessary, change the label to the string resource for the new app name:
4.
```
android:label="@string/app_name"
```

How to delete a project

All the files for an Android project are contained in the project's directory on the computer's file system. To delete a project, delete its directory from your computer's file system.

Android Studio also keeps a list of recent projects that you've opened. You can remove a project from the list of recent projects, but doing so doesn't affect the actual project files. Doing this step is optional.

To remove a project from the list of recent projects in Android Studio, do *one* of the following:

- In the Android Studio welcome screen, place the cursor on the deleted project. Click the **X** that's displayed on it, or press the delete key.
- In the Android Studio welcome screen, click on the project name and select **Remove From List**.
- Select **File > Open Recent > Manage Projects**, select the project, and press the delete key or click the **X** that's displayed on it.

How to extract resources

Strings

For your app to be translatable into multiple languages, you must keep all your string resources in the `res/values/strings.xml` file.

Creating string resources

You can manually add string resources to the `strings.xml` file using the following syntax:
`<string name="string_name">String Value</string>`
To extract a string that you've hardcoded:

1. Select the existing, hardcoded string in either XML or Java.
2. Press `Alt+Enter` (`Option+Enter` on a Mac), or click the orange light bulb to the right of the string.
3. Select **Extract string resource**.
4. The **Extract Resource** window is displayed. Supply a name for **Resource name:** and click **OK**.

Accessing string resources

To reference a string resource:

- In XML, use the following syntax: `@string/string_name`
- In Java, use the following syntax: `getString(R.string.string_name)`

Extracting dimensions

In general, keep dimensions in the `dimens.xml` file instead of hardcoding them. Keeping dimensions in `dimens.xml` allows you to specify different dimensions using resource qualifiers. You extract dimensions the same way you extract strings: press `Alt+Enter` (`Option+Enter` on a Mac), or click the orange light bulb to the right of the dimension.

Extracting styles

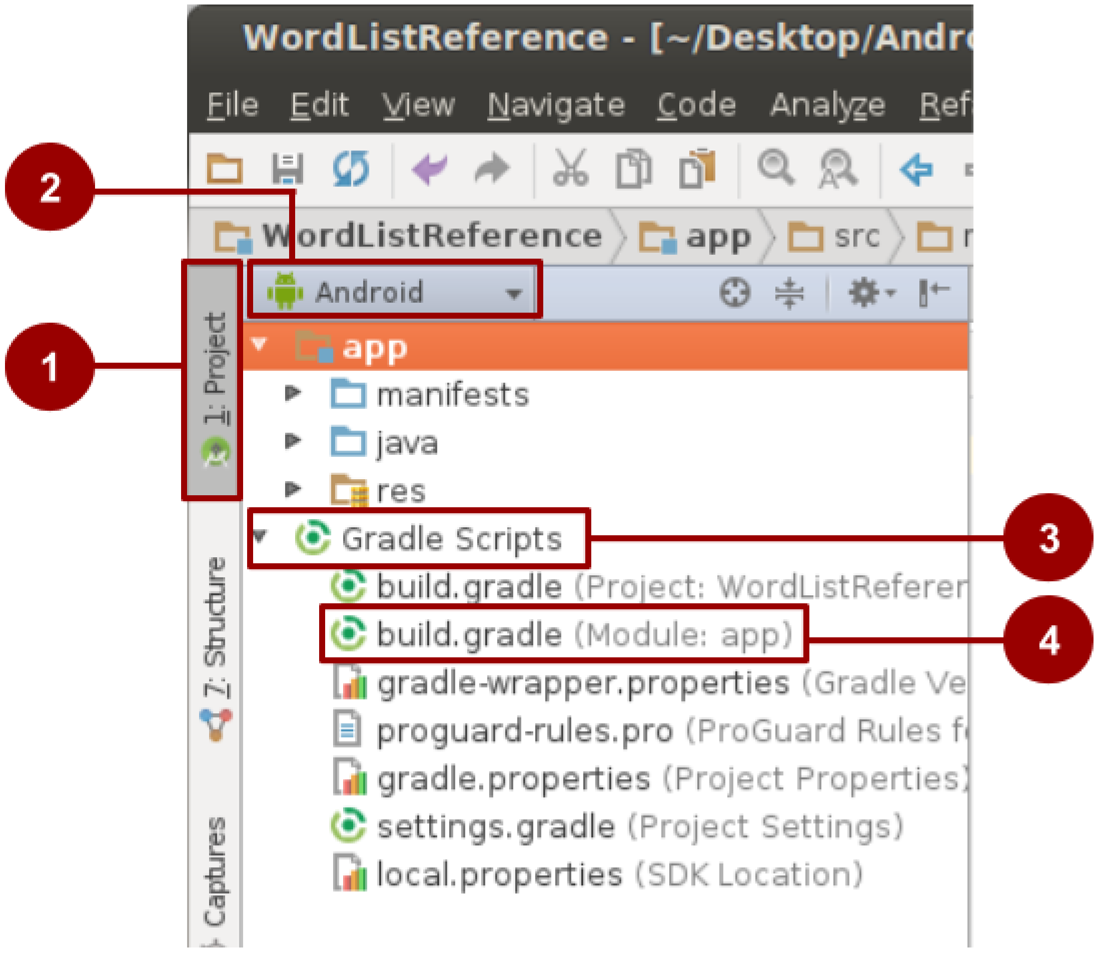
If you have several elements in your app that share the same attributes, you can create a style in the `style.xml` file for those attributes.
To extract existing attributes into a style, do the following:

1. In the layout editor **Design** tab, right-click on the view in the **Component Tree** and select **Refactor** > **Extract Style...**.
2. The **Extract Android Style** window is displayed. Name the style and select attributes. If you want Android Studio to search the rest of the file for the selected attributes and apply the style to views where the attributes match, select **Launch 'Use Style Where Possible' refactoring after the style is extracted**.
3. Click **OK**.

How to add Android support libraries

The Android Support Library provides backward-compatible versions of Android framework APIs, additional UI components, and a set of useful utilities.

For example, the `RecyclerView` class is located in the Android Support package. To use `RecyclerView`, you must include the `RecyclerView` support library dependency in your project's `build.gradle` file. The process is the same for other Android Support Library components.
To add support libraries for `RecyclerView` to your Gradle file, follow these steps and refer to the screenshot below:


WordListReference - [~/Desktop/Andr
File Edit View Navigate Code Analyze Ref
WordListReference app src
Android
1: Project
7: Structure
Captures
app
manifests
java
res
Gradle Scripts
build.gradle (Project: WordListReferen
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Ve
proguard-rules.pro (ProGuard Rules f
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
1
2
3
4

1. In Android Studio, in your project, make sure that you are in the **Project** pane (1 in the screenshot) and in the **Android** view (2).
2. In the hierarchy of files, find the **Gradle Scripts** folder (3).
3. Expand **Gradle Scripts**, if necessary, and open the **build.gradle (Module: app)** file (4).
4. Toward the end of the **build.gradle (Module: app)** file, find the dependencies section. There is probably a line similar to the following one:
5. `implementation 'com.android.support:appcompat-v7:27.1.1'`
6. Below that line, add the following code:
7. `implementation 'com.android.support:recyclerview-v7:27.1.1'`
8. Change the version number of the new line to match the latest version or version number of the existing library.
9. Make sure the version numbers of all the libraries are the same and match up with the `compileSdkVersion` number. If they don't, you will get an error.
10. Sync your Gradle files.
11. Build and run your app.

The following example shows the dependencies section of the `build.gradle` file with support libraries added for `RecyclerView`, if the version number were `27.1.1`.

```
dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   testImplementation 'junit:junit:4.12'
   implementation 'com.android.support:appcompat-v7:27.1.1'
   implementation 'com.android.support:recyclerview-v7:27.1.1'
   implementation 'com.android.support.constraint:constraint-layout:1.1.3'
}
```

How to create images in Asset Studio

Use Image Asset Studio to create image assets for a launcher icon, a notification icon, or action bar icons:

1. Open your app in Android Studio.
2. Right-click the **res** folder of your project and select **New** > **Image Asset** from menu.

The **Asset Studio** window opens. You can create a text icon, choose from available clip art, or add your own custom image.

Custom text icons

Default launcher icons called `ic_launcher` and `ic_launcher_round` come with your project template. To add a custom text icon:

1. Open the Asset Studio, as described above.
2. So that you won't overwrite the `ic_launcher` icon, change the **Name** of the icon to `ic_launcher_text`.
3. In the **Asset Type** row, select **Text**.
4. Enter "Hello World!" into the **Text** field. Experiment with changing the font and the font color.
5. Experiment with changing the background color in the **Background Layer** tab.
6. Click **Next.** The **Confirm Icon Path** window shows how an icon with your specified text will be created for each resolution.
7. Click **Finish**. The **res/mipmap** folder now contains the new icons.

To use the new icon:

1. Open the `AndroidManifest.xml` file. Change the `android:icon` line so that instead of referencing `ic_launcher_round`, it references `ic_launcher_text_round`:
2.
```
android:icon="@mipmap/ic_launcher_text_round"
```
3. Run your app.
4. After the app has launched, go to the home screen and open the list of apps.
5. Scroll to find the new app icon listed along with the other installed apps.

Clip art icons

To add a clip art icon, follow the steps to add a custom text icon, with the following changes:

1. Set the **Name** of the icon to `ic_launcher_clipart`.
2. Choose **Clip Art** as the **Asset Type**.
3. In the **Clip Art** row, click the button that shows the current icon, the default Android.
4. Select an icon from the **Select Icon** dialog.

Custom icons

To add a custom icon, follow the steps to add a custom text icon, with the following changes:

1. Set the **Name** of the icon to `ic_launcher_image`.
2. Choose **Image** as the **Asset Type**.
3. In the **Path** field, choose an image. The image can be one that you've added to your project, or an image on your computer.

How to compare custom objects

If your data model calls for objects to be sorted, you need to define how the objects can be compared to each other. To specify how to compare two objects and determine whether one object is bigger, smaller, or the same as the other object, use the `Comparable` interface.
The `Comparable` interface requires you to implement a single method: `compareTo(<T> another)`, where is the parameterized type you implemented `Comparable` with, and the type of object you are comparing to.
For example, to compare your `Foobar` instance to other `Foobar` instances, you implement `Comparable<Foobar>`. Your `compareTo()` method takes `Foobar` as a parameter.
The `compareTo()` method should do the following:

- Return a negative integer if the object is less than the parameter.
- Return a positive integer if the object is greater than the parameter.
- Return zero if the objects are equal.

For example, to compare a list of books by publication date, you could use the following code:

```
@Override
public int compareTo(Book book) {
    if (this.publication == book.publication) { return 0; }
    else { return this.publication > book.publication ? 1 : -1;}
}
```

How to save the state of custom objects

In Android, you often create custom objects to represent your data model. To preserve the state of custom objects, you must be able to pass them into the `savedInstanceState` bundle, and therefore your custom class must implement the `Parcelable`interface.
The `Parcelable` implementation allows for primitive types (`int`, `string`, `byte`, etc.) to be saved in the `savedInstanceState` callback.
To save the state of custom objects, follow these steps:

1. Set up the data in your custom class.
2. Add the `Parcelable` implementation to your class declaration. Remember that only the primitive data types will be saved.
3. In Android Studio, the class declaration is underlined in red, because you need to implement the interface methods. With your cursor on the underlined text, press `Alt+Enter` (`Option+Enter` on a Mac) and select **Implement methods**.
4. Select `describeContents()` and `writeToParcel(Parcel dest, int flags)`.
5. Click **OK.** The class name is still underlined, indicating that the interface is not fully implemented yet.
6. Select the class name and press `Alt+Enter` (`Option+Enter` on a Mac) and select **Add Parcelable implementation**. Android Studio adds the required code. The variables for which you want to preserve the state (primitive types) are written to the `Parcel` in the `writeToParcel` method.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2

You can now save the state of these objects using the `savedInstanceState` bundles methods: `putParcelable()`, `putParcelableArray()`, `putParcelableArrayList()`, and the respective "getter" methods.