UNIVERSITY OF ALBERTA
FACULTY OF SCIENCE
Department of Computing Science

SOFTWARE DESIGN AND ARCHITECTURE

Android Studio Tips: A how-to / what-is / tip guide

Table of Content

This guide is meant to give you quick access to information on a variety of Android Studio topics. The topics covered in this guide are:

In the sections below each topic is briefly introduced and you will learn how to perform common or useful Android Studio tasks related to that topic.

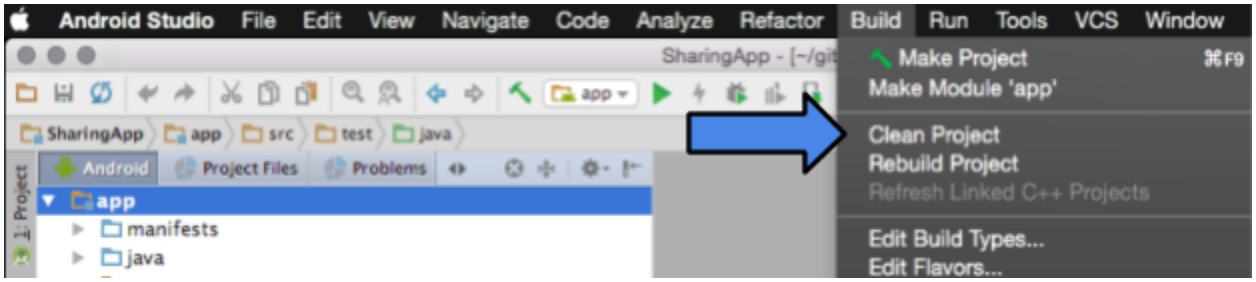
1. Clean the Project

When you run your app for the first time, Android Studio compiles the project and saves the resulting build artifacts to the Build folder.

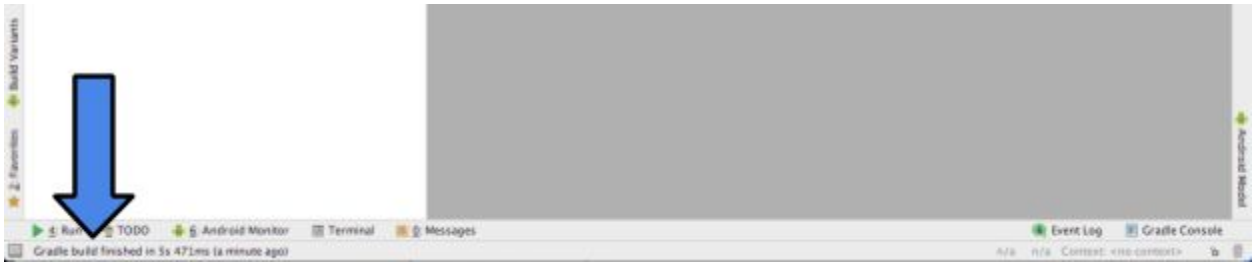
If you make major changes to the code base, you may want to recompile the app before you run it again. If you don't, Android studio may run the previous build -- and as a result, you may encounter errors that don't make sense because they refer to a previous version of the code.

To prevent this frustration, after you've made a major change to the code base, clean the project before running the app. Cleaning the project removes the artifacts in the Build folder which forces the app to be recompiled before it is run again.

To clean the project, click **Build**, then **Clean Project**.



This may take a few minutes to complete. When this process is complete you will see confirmation in the bottom left of the Android Studio window, which will say "Gradle build finished".



2. Clear the App's Data

When you make a major change to the type of data stored in your app, the app may crash the next time you run it. To help understand why this may happen, let's look at an example.

> Let's assume you have previously run SharingApp and added an item to your inventory and have set it's status to borrowed. In this version of the app the borrower is stored as type String.
>
> You now decide you want to update the application so that a borrower is stored as a Contact, not a String. You update the borrower attribute in the Item class and refactor the rest of the code base to now reflect that the borrower attribute is of type Contact.

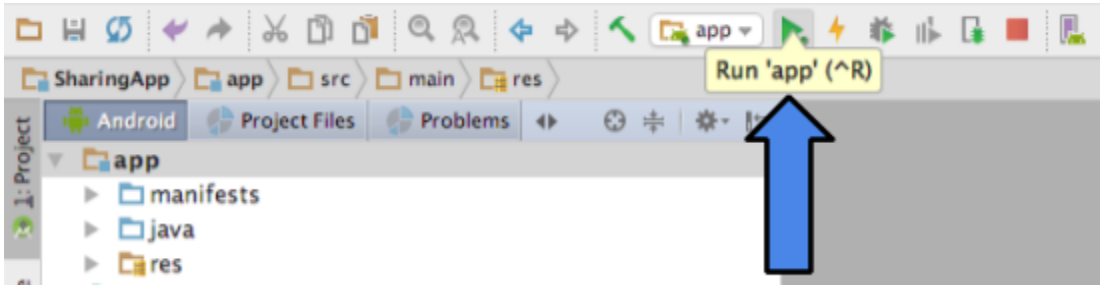
Everything seems fine, you would expect the app to work… but when you run it, it immediately crashes.

The app crashes because it is trying to load the previously stored item -- but the previously stored item is not the same type of item anymore, since the now the Item class has a borrower attribute of type Contact.

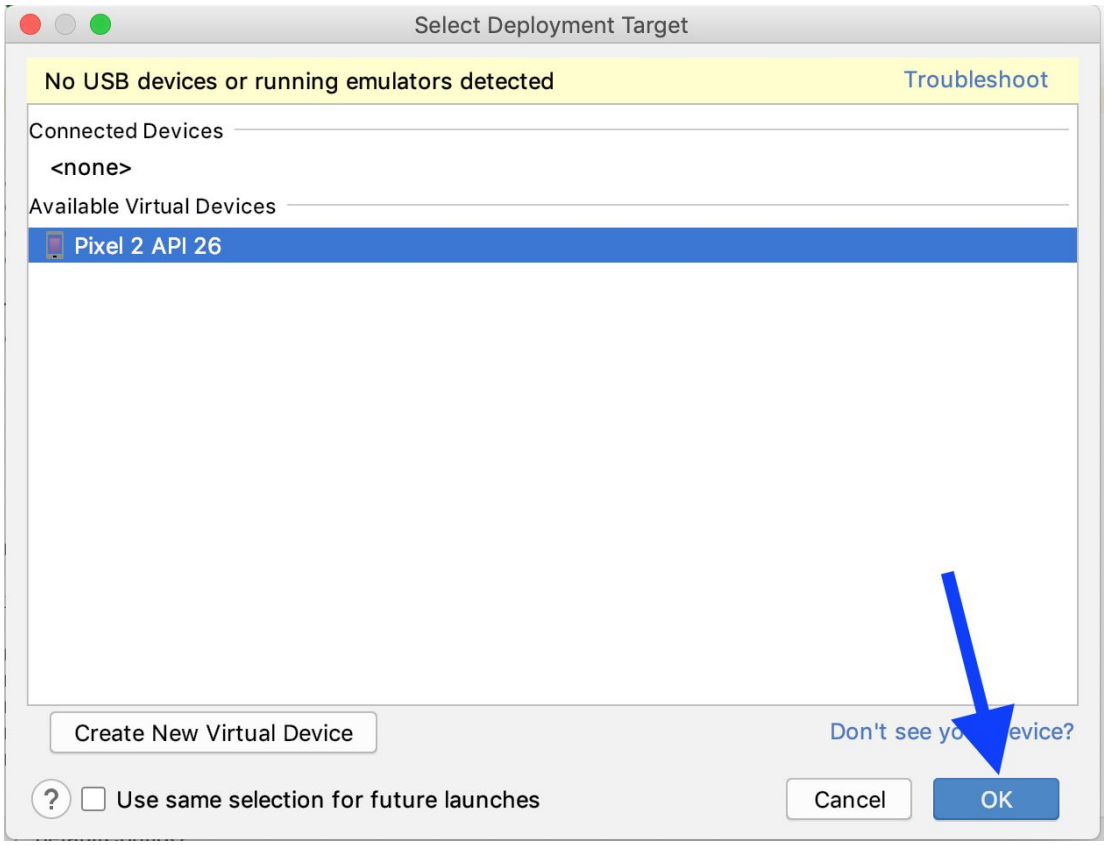
As long as the old item is stored in the app's local data, the app will crash when it tries to load the previously stored data.

To get around this error you will need to delete all previously saved data in the app. To delete the locally stored data you must do the following:

Click the **play button** to run the app.



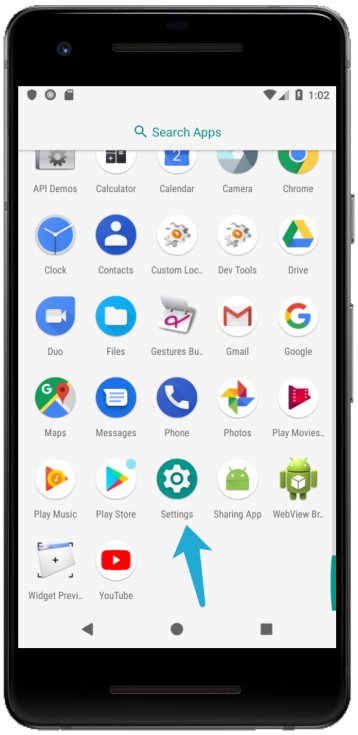
Select the emulator from the list and click **OK**.

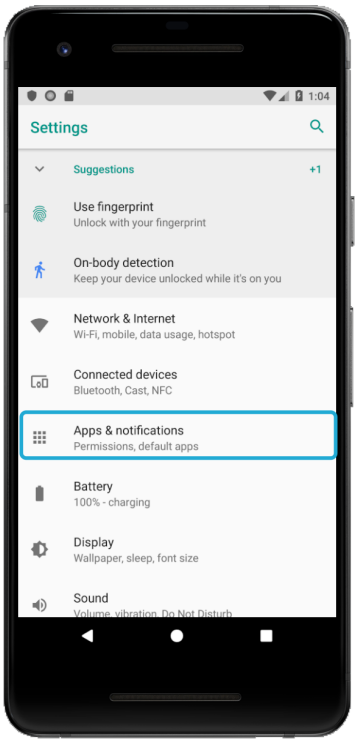


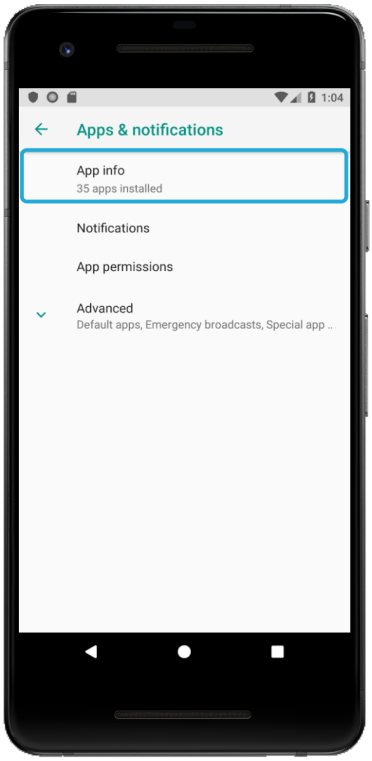
Be patient, the emulator may take a few minutes to load.
If the app launches and doesn't crash -- great! You are done. Apparently the changes you made to the app did not have an effect on the data being stored.

If it does crash -- don't worry. A message will appear to inform you that the app has crashed. Click **OK**. Then, click the button near the bottom of the screen that is made up of six circles.

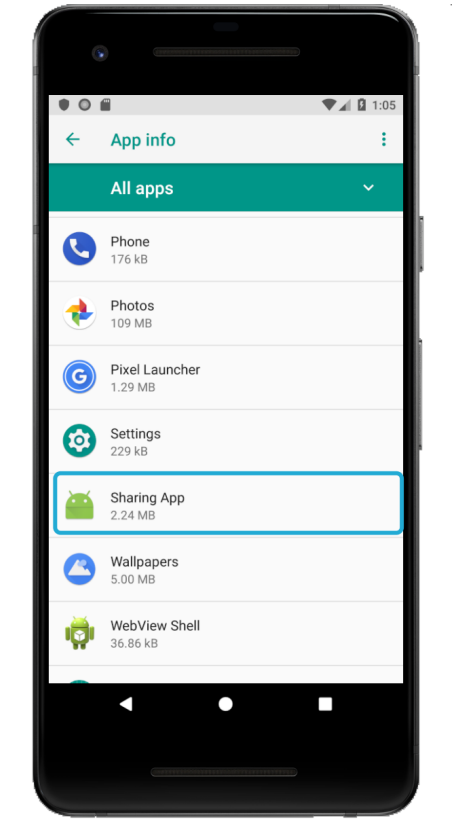
Click and drag to scroll through the apps until you find the **Settings** app. Click **Settings**. Then click **Apps and notification** -> **App Info**







This displays all apps on the emulator. Click and drag to scroll through the list. Near the bottom of the list you will find **Sharing App**. Click **Sharing App**.



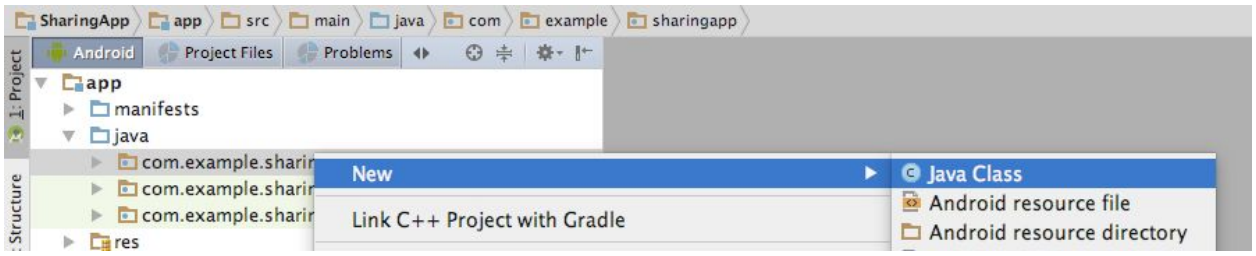
After clicking **Sharing App**, click **Storage**. Then click **CLEAR DATA**. A message will pop up asking you to confirm this action. Click **OK**.

Now all the previously stored data has been erased. The next time you run your app it shouldn't crash… unless you have a different error.

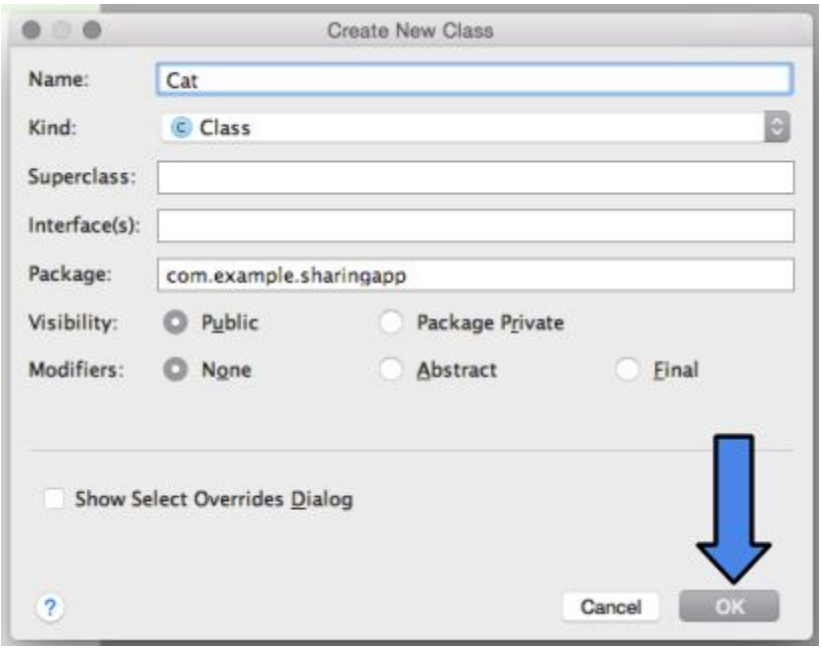
---

3. Make a New Class

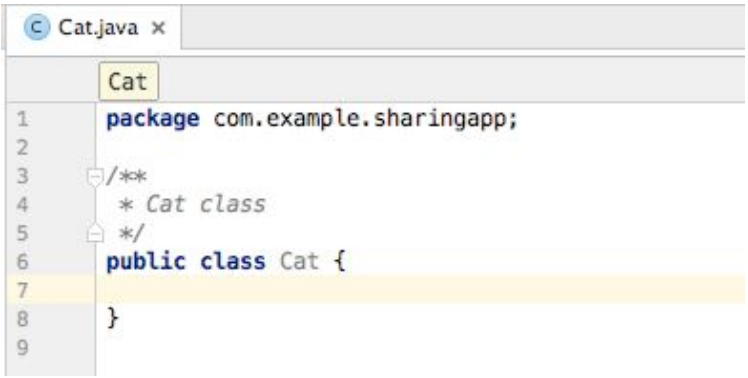
You can create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New → Java Class**.



For example, you could name the new class **Cat**. Click **OK**.



This creates an empty **Cat** class which you can then implement.

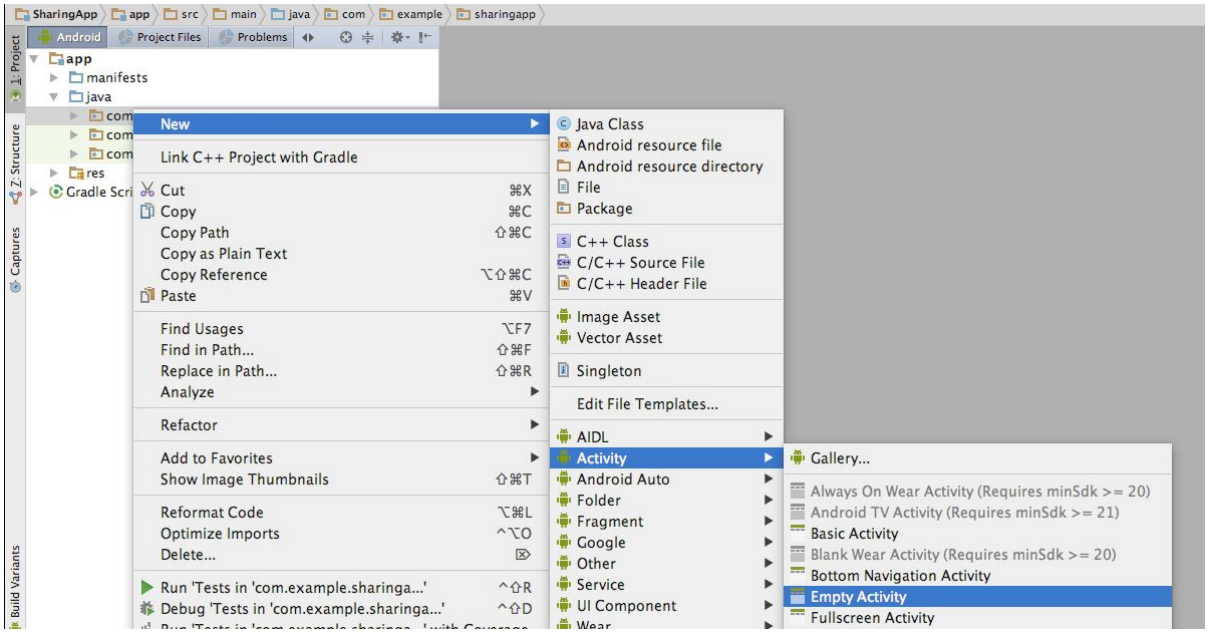


---

4. Make a New Activity

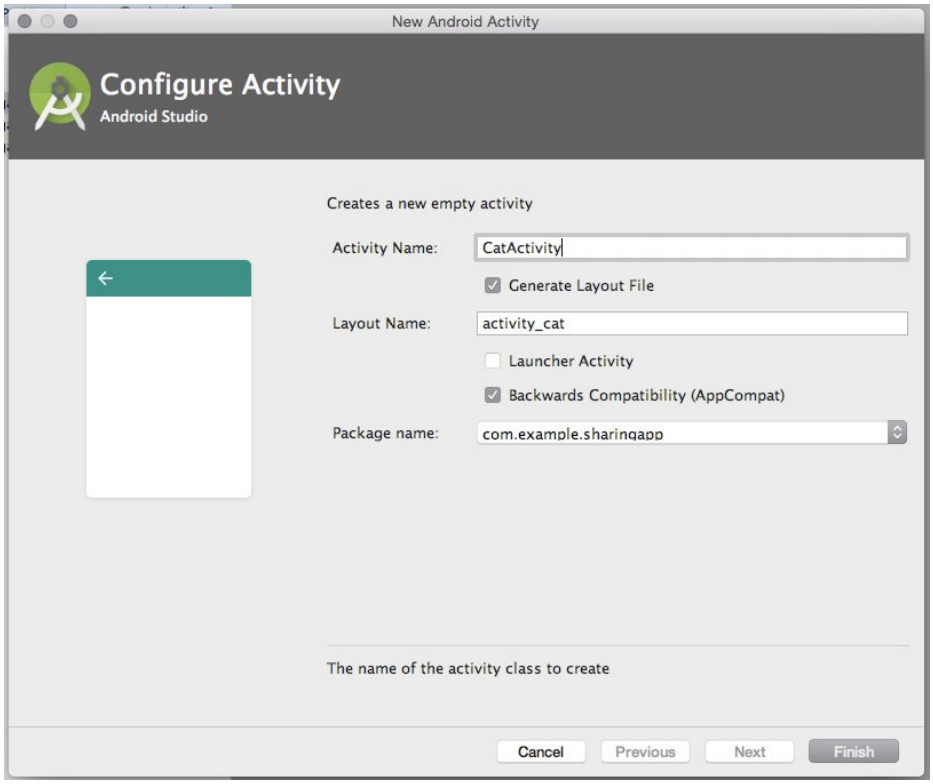
For the purposes of our Capstone assignments, when we create a new activity we always want to be an **Empty Activity**.

To create a new activity right click on the **com.example.sharingapp** folder, then click **New** → **Activity** → **Empty Activity**.



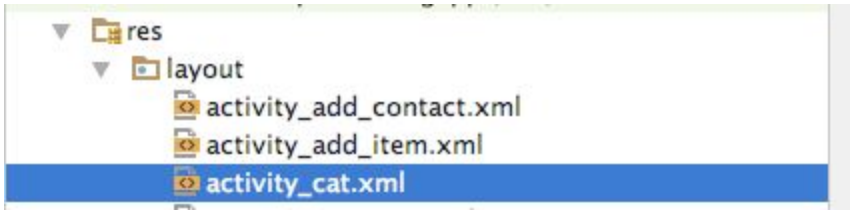
For example, you could name the new activity **CatActivity** and the resource file **activity_cat**. Then click **Finish**.

Note: It's a good idea to name your activities using the convention <description>Activity and the corresponding resource files activity_<description>. Follow this naming convention to keep your project more organized!

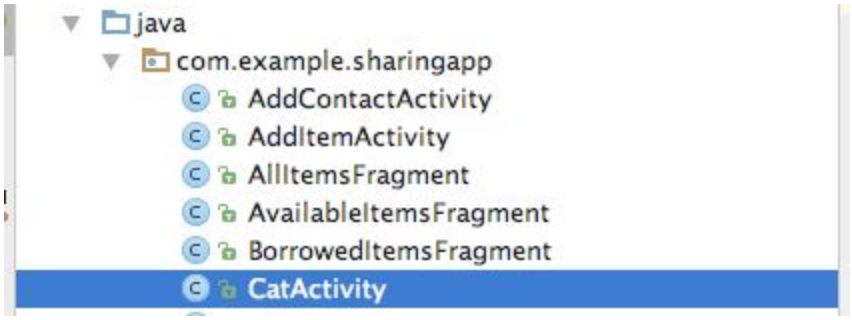


As a result:

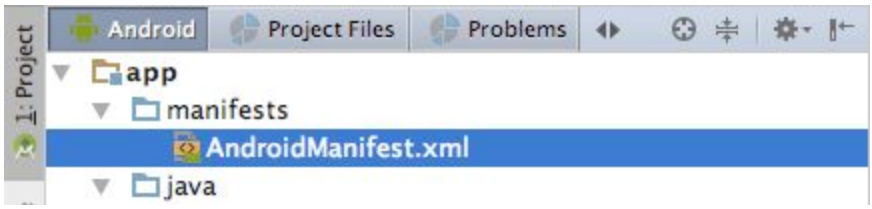
- A new layout resource in the **layout** folder called **activity_cat.xml** is created.



- A new activity class called **CatActivity** is created.



- And the line:
  <**activity android:name=".CatActivity**">
  Is added to the **AndroidManifest.xml** file to link **CatActivity** to all the other activities in the app.



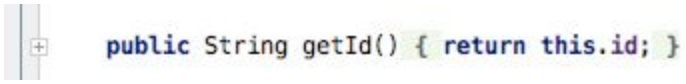
If you want to learn more about Android Activities this is a good resource:
https://developer.android.com/guide/components/activities/intro-activities.html

---

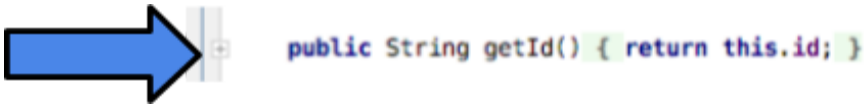
5. Code Folding

Android Studio by default will collapses certain lines of code. This collapsing is referred to as “code folding”. Code folding is not unique to Android Studio, but a common feature of IDEs.

Folded code will have a **+** next to it and will have green brackets.



You can unfold code by clicking on the **+** next to it.



Or, you can unfold the code by clicking on one of the green highlighted brackets.

After you have expanded (or unfolded) the code you will now notice that the full code is revealed and there are now - next to it.

```
public String getId(){
    return this.id;
}
```

You can refold the code by clicking on either of the - next to it.

---

6. How to Import Packages

Android Studio will inform you when you are missing a required package by showing the package dependent code in red.

For the sake of learning, let's take a look at what happens when we delete an import statement.

> For example, when we delete the UUID import statement from the top of the Item class, the UUID related code in the file will be shown in red.
>
> ```
> public void setId() {
>     this.id = UUID.randomUUID().toString();
> }
> ```
>
> When you hover over UUID it gives you an error message, "Cannot resolve symbol 'UUID' ".
>
> To fix this you need to import Java support for UUIDs. To do this, click the red text and press **alt** and **enter** at the same time.This adds the following import statement to the top of the class:
>
> **import** java.util.UUID;
>
> (This is the import statement we previously deleted)

Normally you wouldn't go deleting import statements, however, when adding code to your project "Cannot resolve symbol" errors are common, so you will need to import packages often.

Sometimes it is not obvious to Android Studio which package you would like to import. In this case, you may be asked to select the package from a list of possible packages. Generally, the first package in the list is the one you want.

7. Search: Using the *Find* tool and *Find Usages* tool

It is often desirable to search for a string in the project. For example, you may want to find all the usages of a variable within a class, or see all the usages of a method within the project. Depending on the scope of your search, you may use either the *Find Usages* tool or the *Find* tool.

The *Find* tool

This tool finds all occurrences of a string within a file.

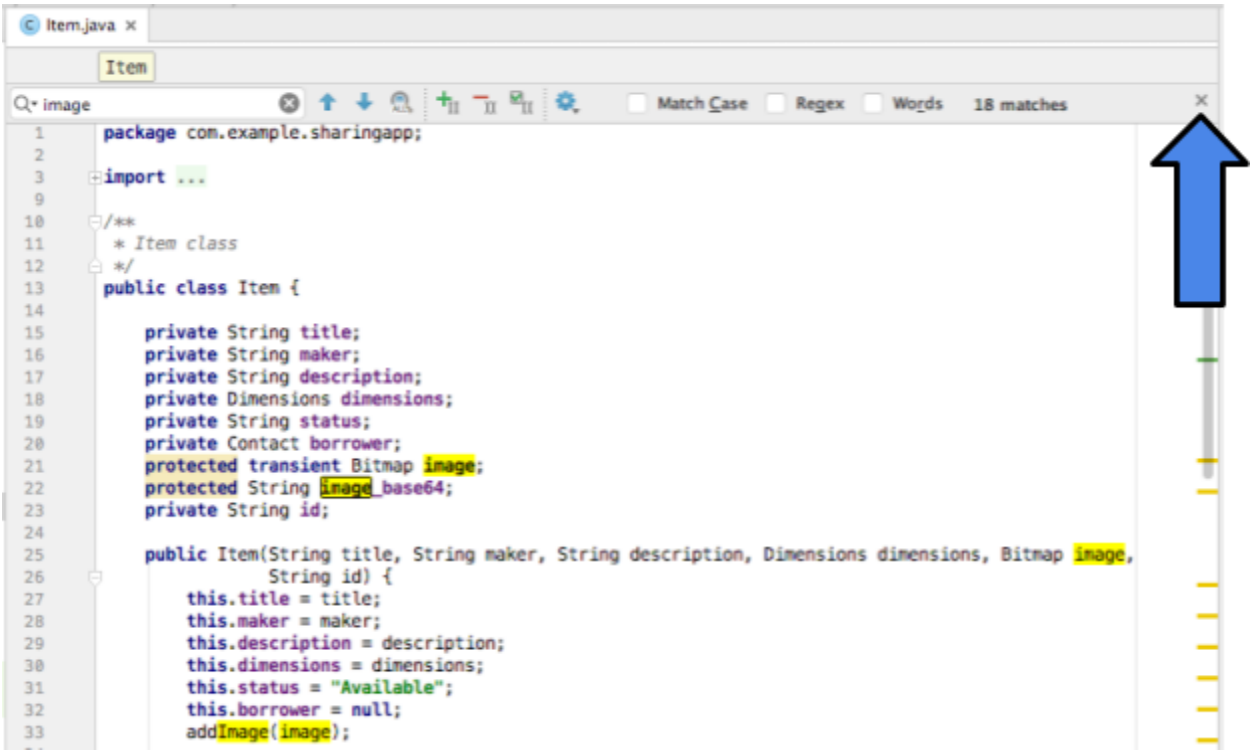
To use this tool, you must first open the file you wish to search in. Next depending on your OS you can press:

⌘ f (on Mac), or
Ctrl f (on Windows/Linux)

A search bar will appear in which you can enter the string you wish to search for.

All occurrences of the string in the file will be highlighted and you can jump to the next occurrence by pressing enter.

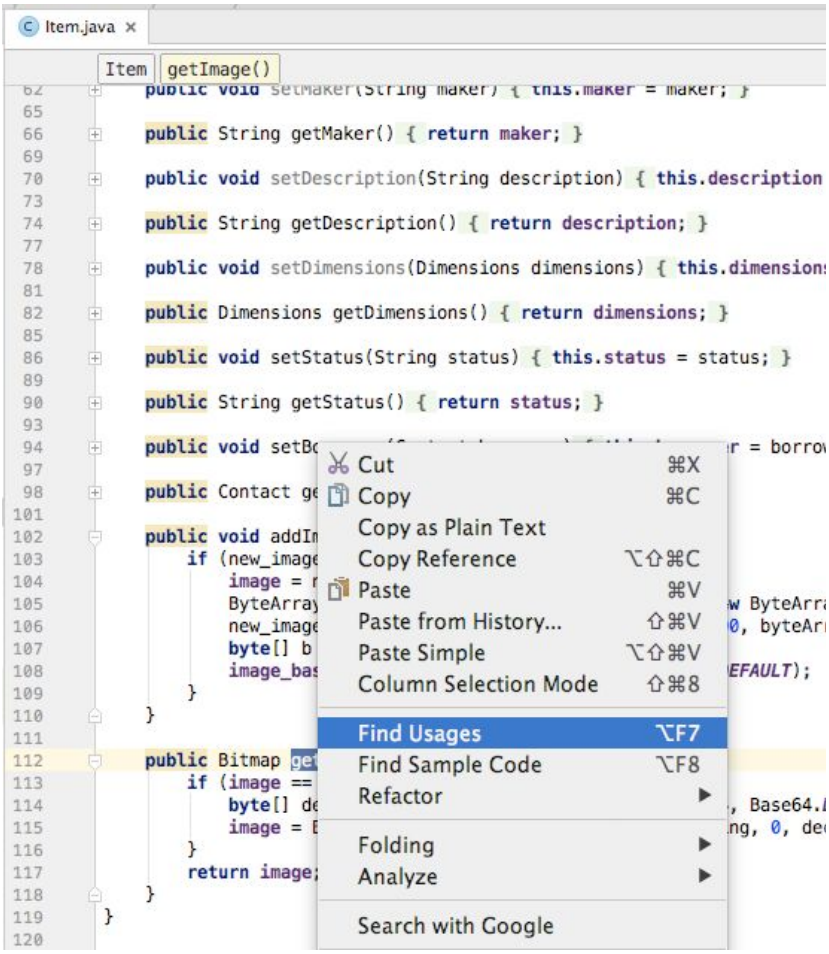
For example, we can search for the string **image** in the Item.java file. When done searching, you can close the search bar by clicking the **x** in the top right corner.



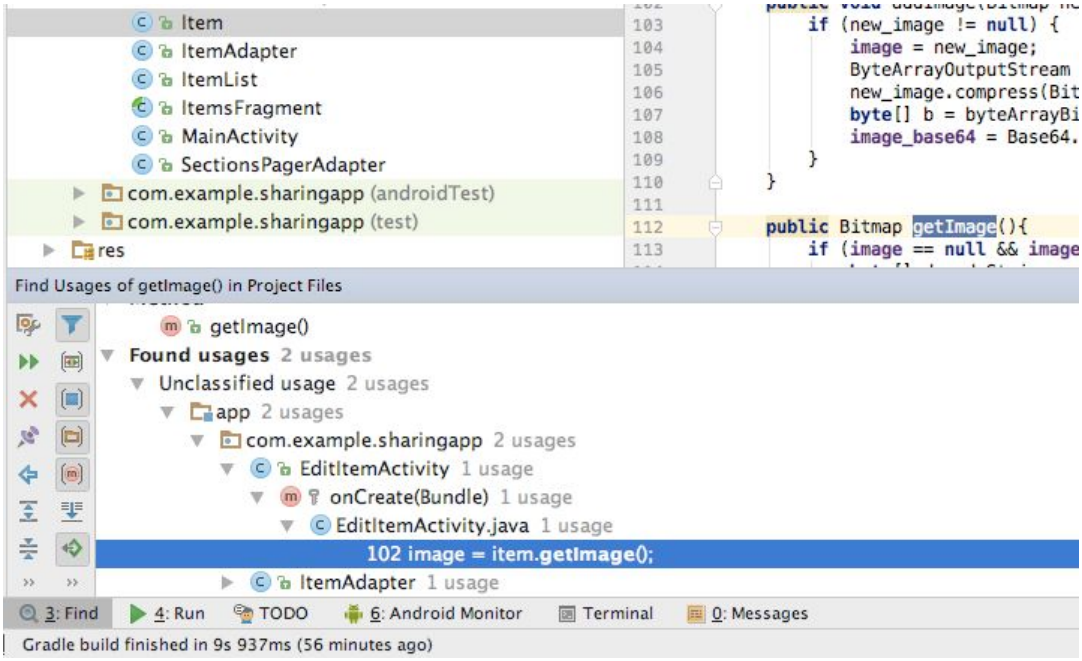
Find Usages

This tool finds all occurrences of a string within the project.

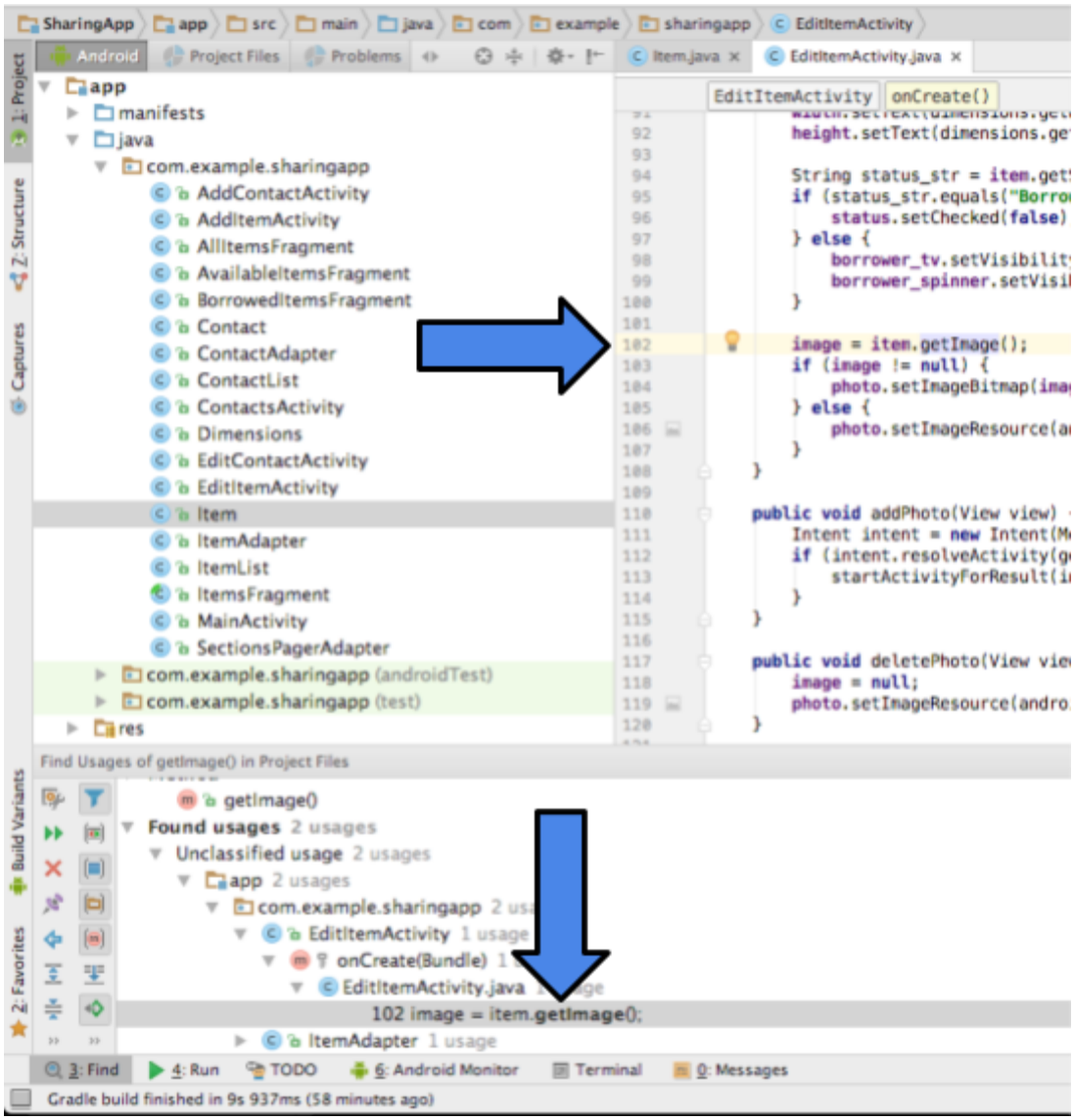
To use this tool you must first highlight the string you wish to search for.
Then right click the highlighted string and click **Find Usages**. For example, we could find the usages of the **getImage** string within the project.



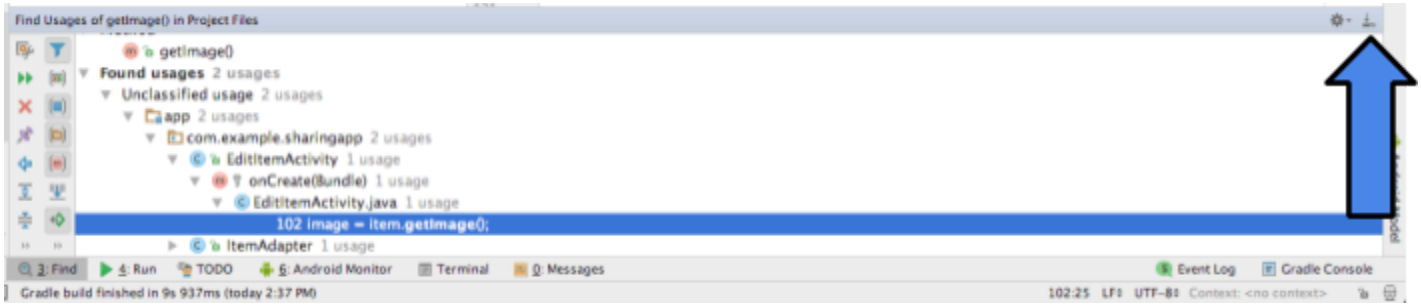
After clicking **Find Usages**, the results of the search will pop up near the bottom of the window:



You can double click on the usage preview line (102 image = item.getImage();) to open that file (**EditItemActivity.java**) and jump to that line in the file.



When done searching, you can minimize the search results by clicking the **hide icon** in the right corner of the search result window.

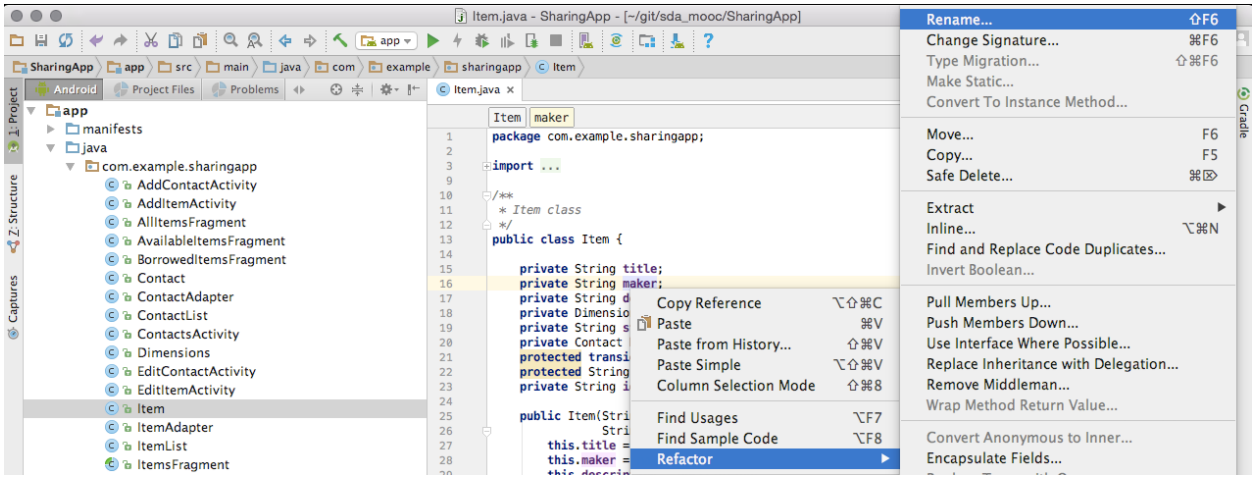


8. The *Rename* tool

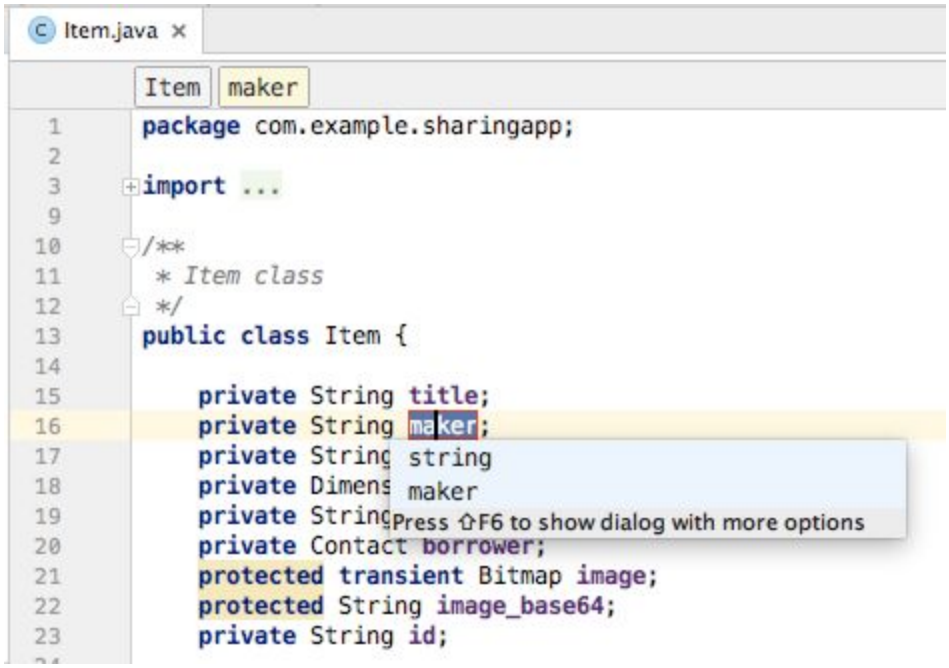
Sometimes you'll want to rename an attribute, class or method. There is no need to manually refactor. You can use the rename tool to make renaming painless.

For example, let's assume that you want to rename the **maker** attribute of the Item class to be **brand** instead.

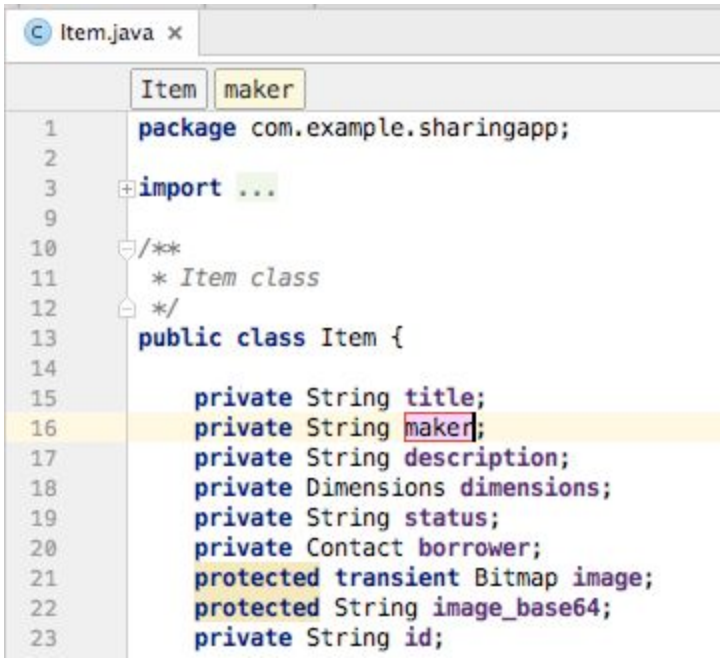
To do this, open the **Item** class and right click on **maker**.



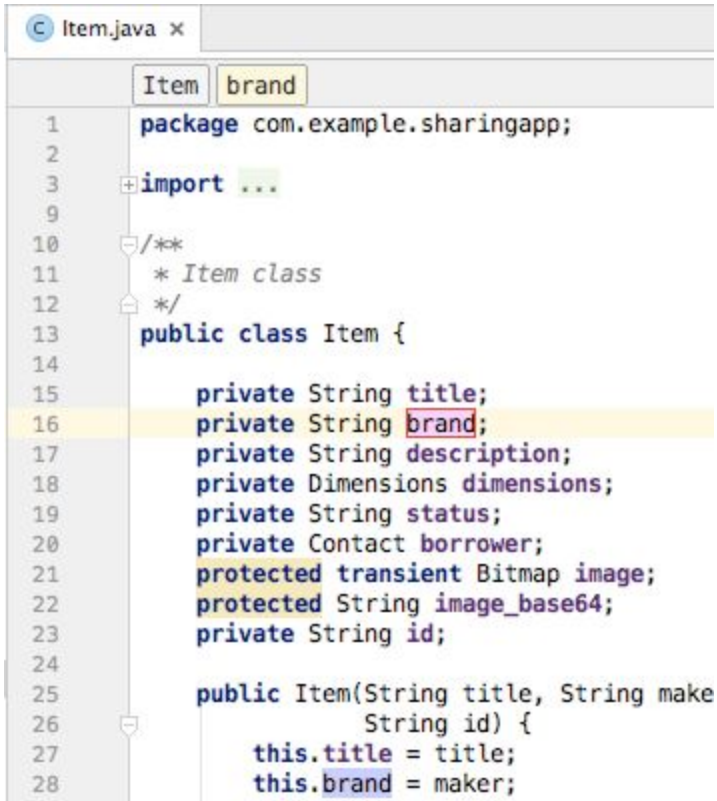
The maker field in the file will now appear in a red box. Some suggested names are listed.



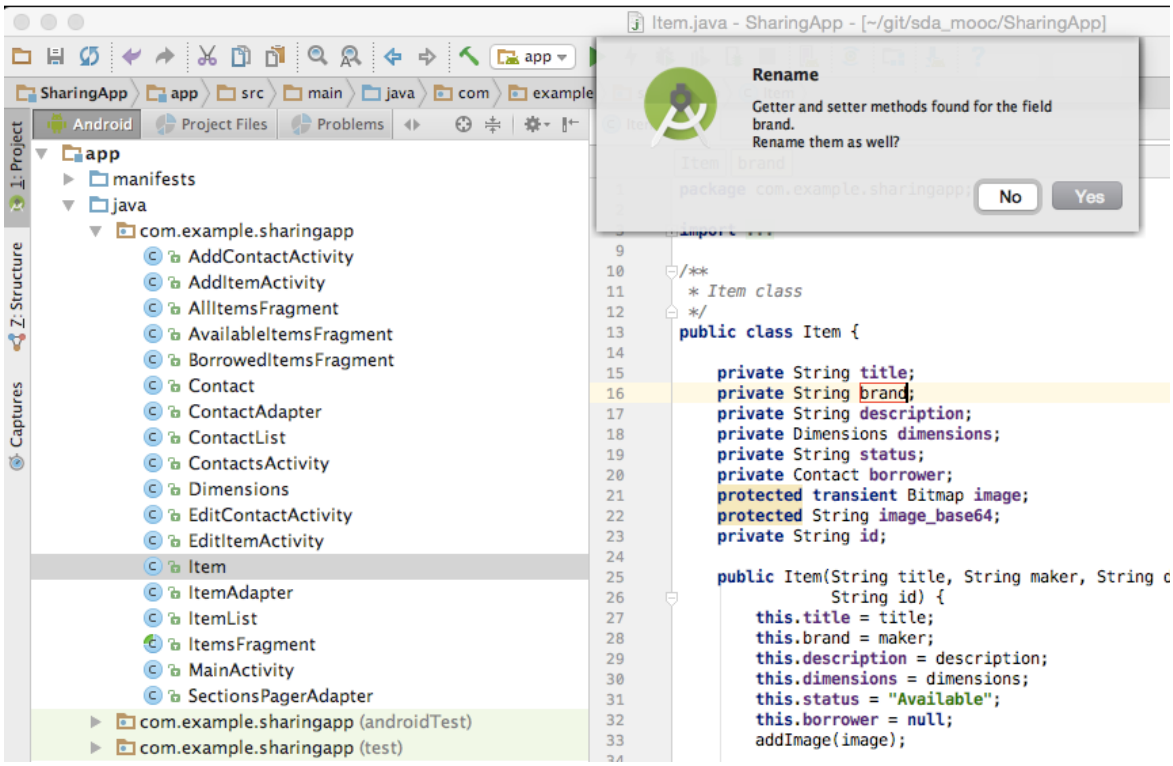
Click after the last letter in the box to move the text cursor to the end of the word:



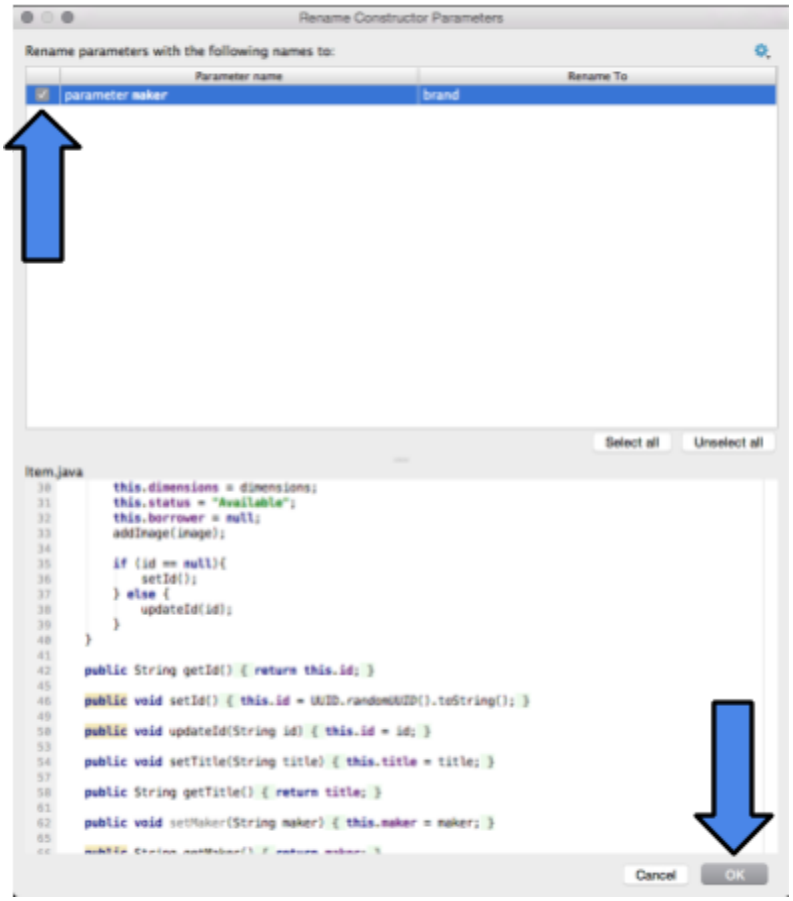
And then delete the word **maker** completely (by repeatedly pressing **backspace**). Next type the new name, **brand**. Press **enter** when you have typed out the entire word.



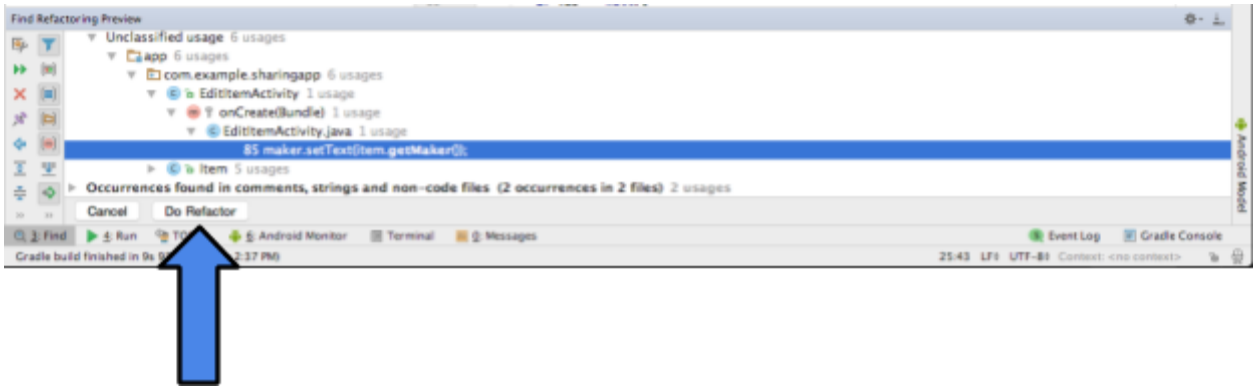
For this refactor, since there are getter and setter methods that contain the word **maker**, a window pops up asking us if we want to refactor these as well. We do, so click **Yes**.



Next, because there is a constructor that has a **maker** parameter, another window pops up that asks us if we want to update this. We do. Select the **checkbox**, then click **OK** to make this change.



Finally, a Refactor Preview will appear near the bottom of the window. Click **Do Refactor** to complete the refactor.

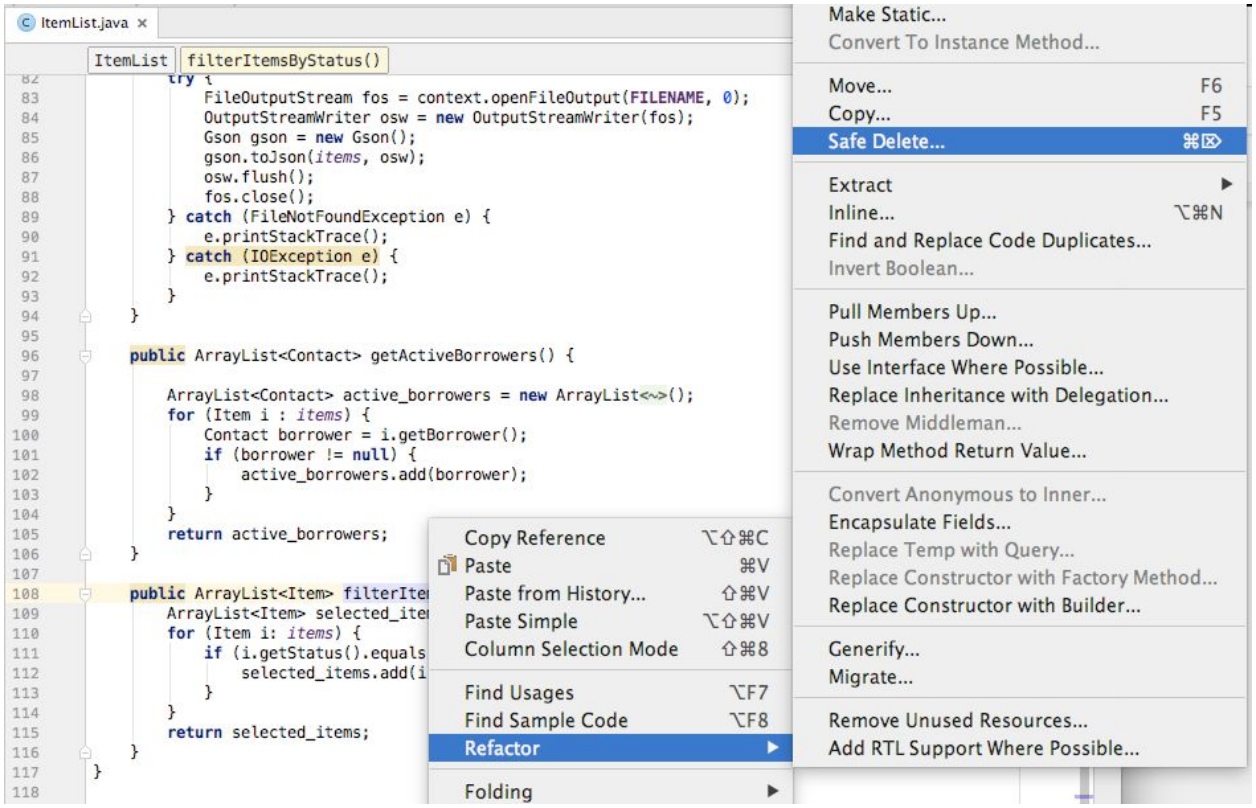


---

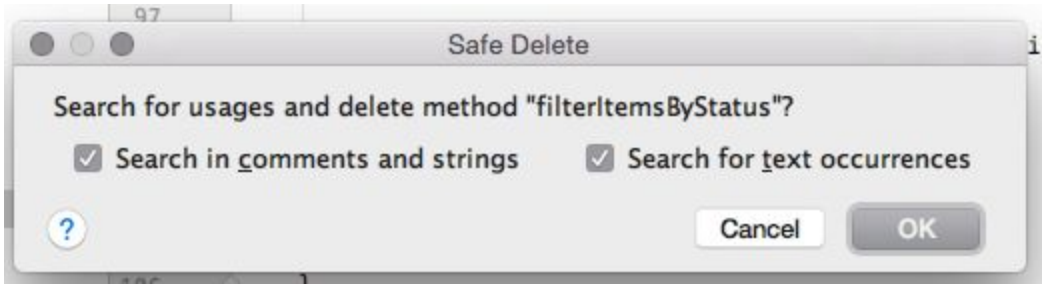
9. How to Delete Classes and Methods using the *Safe Delete* tool

So you need to delete something, but what if other code uses it?
Safe Delete not only allows you to delete classes and methods, but it also warns you about possible issues that result from the deletion.

You can safely delete a method by right clicking on it, then click **Refactor → Safe Delete**.

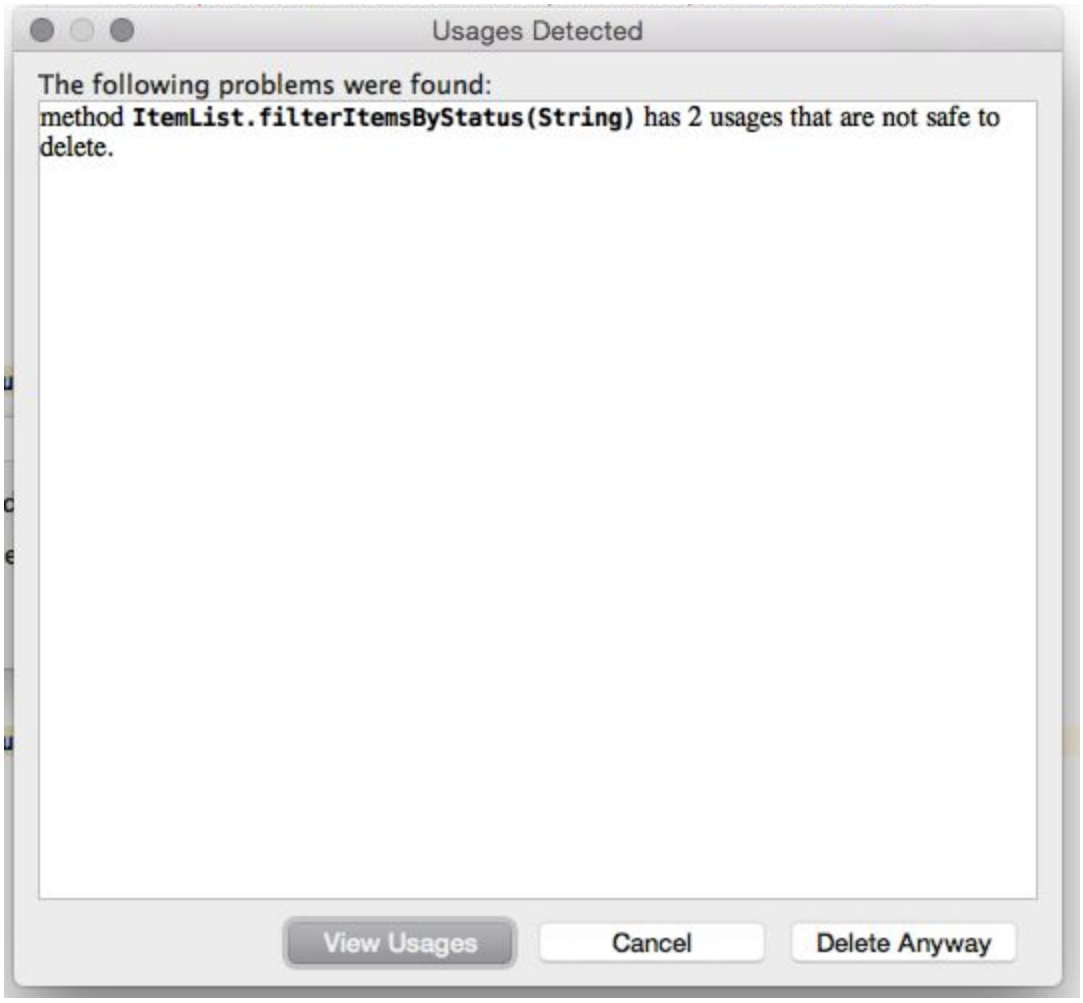


A message will popup to confirm that you would like to search for all usages of the method before deleting it. Click **OK**.



Next, if any potential issues were found, they will be reported.

Depending on what you are trying to accomplish, you may want to reconsider deleting the method (in which case you can click to **View Usages**, or **Cancel**), or if you want to go ahead with the deletion anyway, you can click **Delete Anyway**.



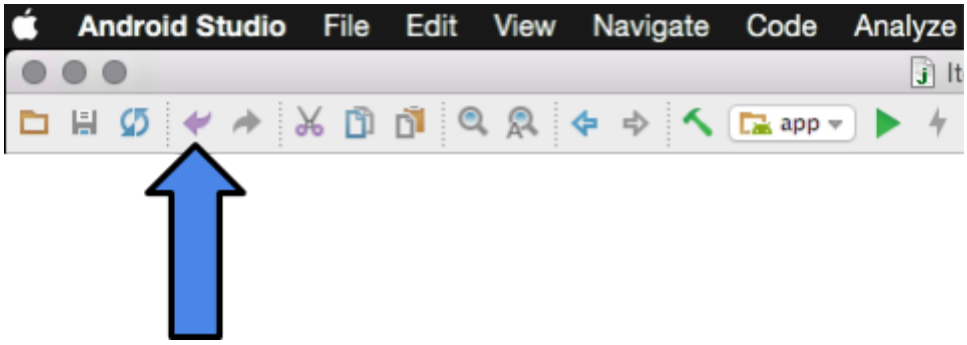
If you do click **Delete Anyway**, then all trace of this method will be deleted from your project.

Similarly, the process of deleting a class is completely analogous to the process of deleting a method.

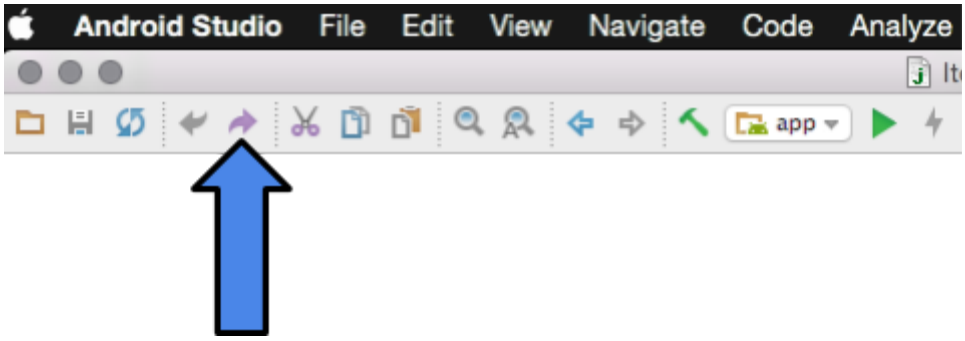
10. How to Undo/Redo

You don't need to fear making changes to your code because you can always Undo and Redo changes.

To **Undo** a change press the undo button



To **Redo** a change press the redo button



The undo and redo buttons will only be highlighted and clickable for the most recent change.

If you want to undo/redo by more than one step you can repeatedly press the following shortcut keys at the same time:

Undo
⌘ z (on Mac)
Ctrl z (on Windows/Linux)

Redo
⌘ shift z (on Mac)
Ctrl shift z (on Windows/Linux)

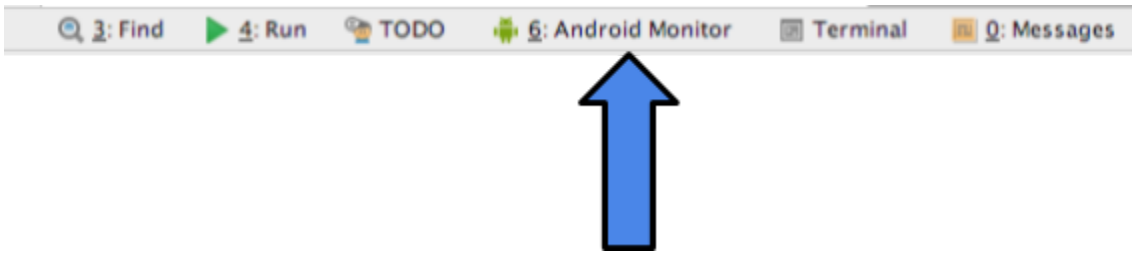
11. Debugging using *Android Monitor*

Android Monitor is a useful tool for monitoring the execution of your app.

The monitor displays runtime errors as they occur on the emulator and various kinds of log information.

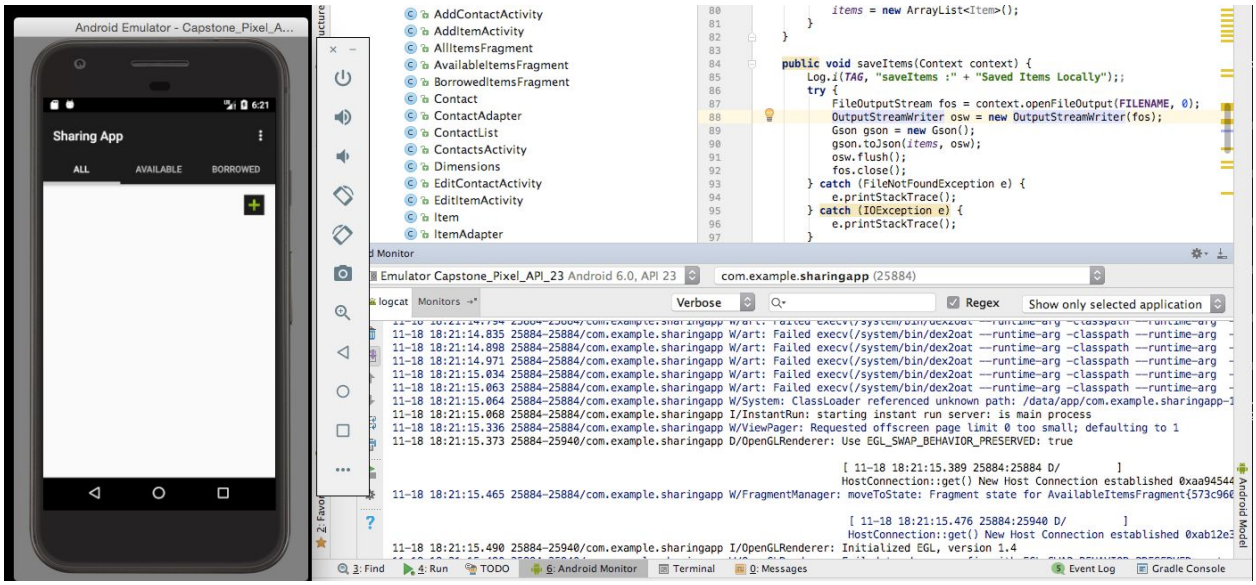
When you add log statements to your code, they appear in the Android Monitor as the application is run.

Open the Android Monitor by clicking on it. (It's located near the bottom left corner of the IDE)



Run the app.

The Android monitor will produce output that looks something like this.



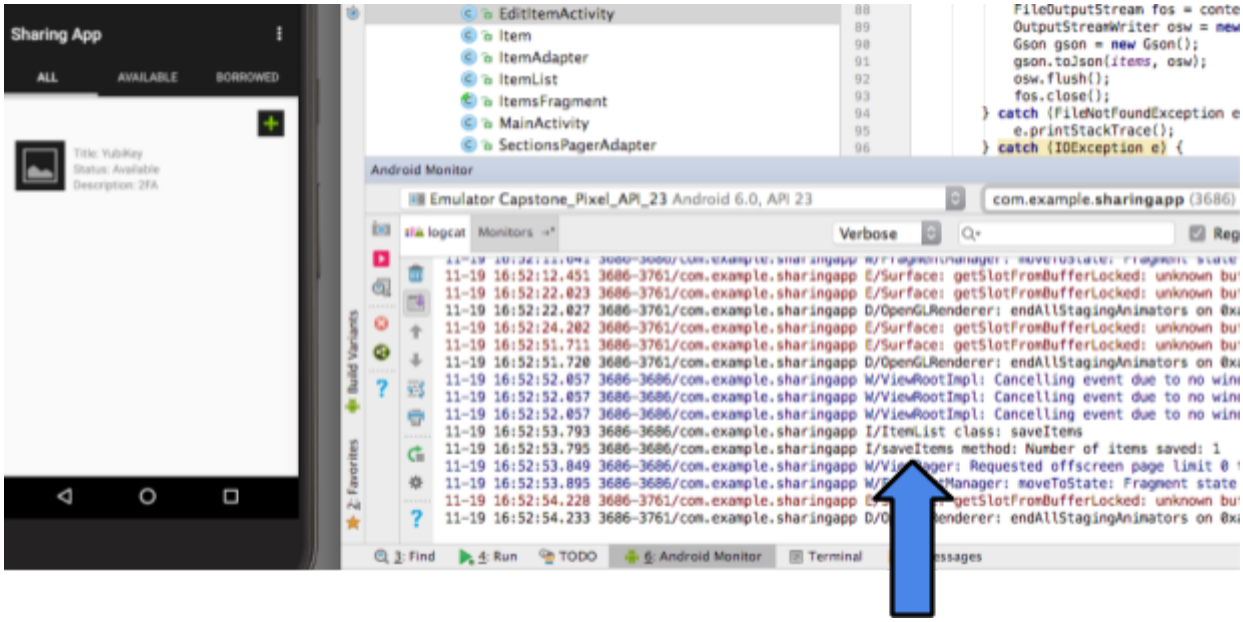
We can use Android Monitor as a debugging tool, by adding log statements to our code.

For example we can add a log statement to the **savelems()** method in the **ItemList** class to see when data is saved locally in the app and how many items have been saved.

```
Log.i("ItemList class", "saveItems");
Log.d("saveItems method", "Number of items saved: " + items.size());
```

```
public void saveItems(Context context) {
    Log.i("ItemList class", "saveItems");
    Log.d("saveItems method", "Number of items saved: " + items.size());
    try {
        FileOutputStream fos = context.openFileOutput(FILENAME, 0);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        Gson gson = new Gson();
        gson.toJson(items, osw);
        osw.flush();
        fos.close();
    } catch (FileNotFoundException e) {
```

Don't forget to save and run the app again after adding these log statements.
As you save a new item to your inventory the following appears in the Android Monitor



More information about how to write log statements can be found here:
https://developer.android.com/studio/debug/am-logcat.html#WriteLogs

---

12. When in Doubt: Restart Android Studio

"It's a feature you would seldom use unless your file caches went ballistic (and that *can* happen every now and then)." - Makoto

Occasionally, you may encounter an error that makes absolutely no sense. After ruling out all obvious solutions it's probably worth restarting Android Studio -- to rule out that Android Studio itself is the cause of the issue.

---

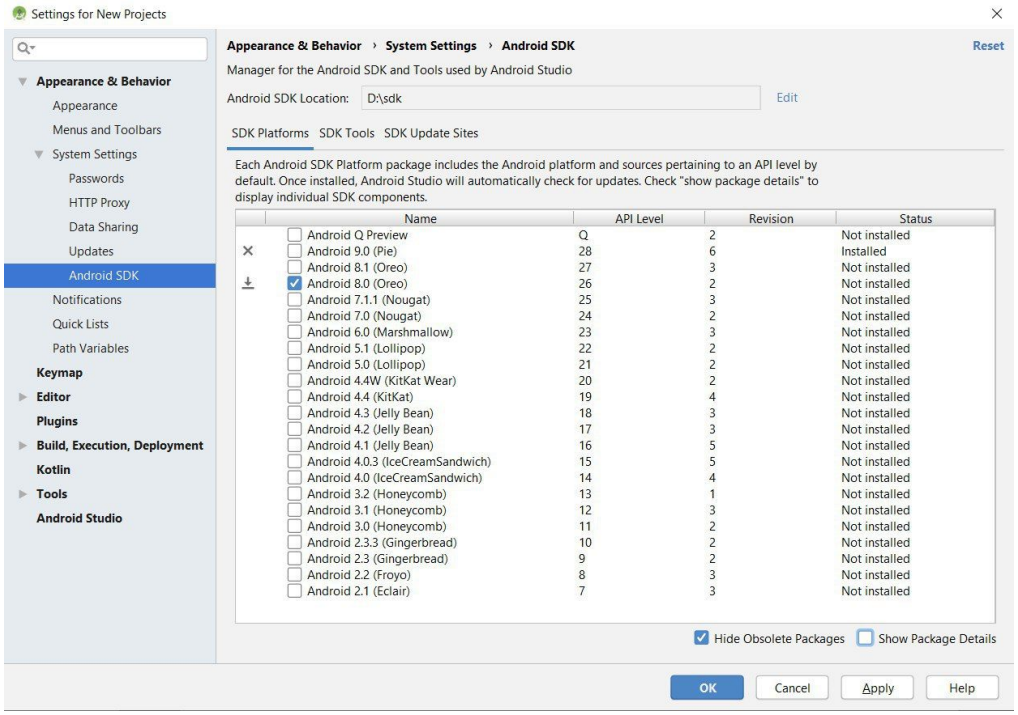
13. Shortcut Keys

There's a lot of buttons and menus in the Android Studio IDE to do all the things you want, but maybe you'd prefer to use shortcuts instead?
View some common shortcuts here: https://developer.android.com/studio/intro/keyboard-shortcuts.html

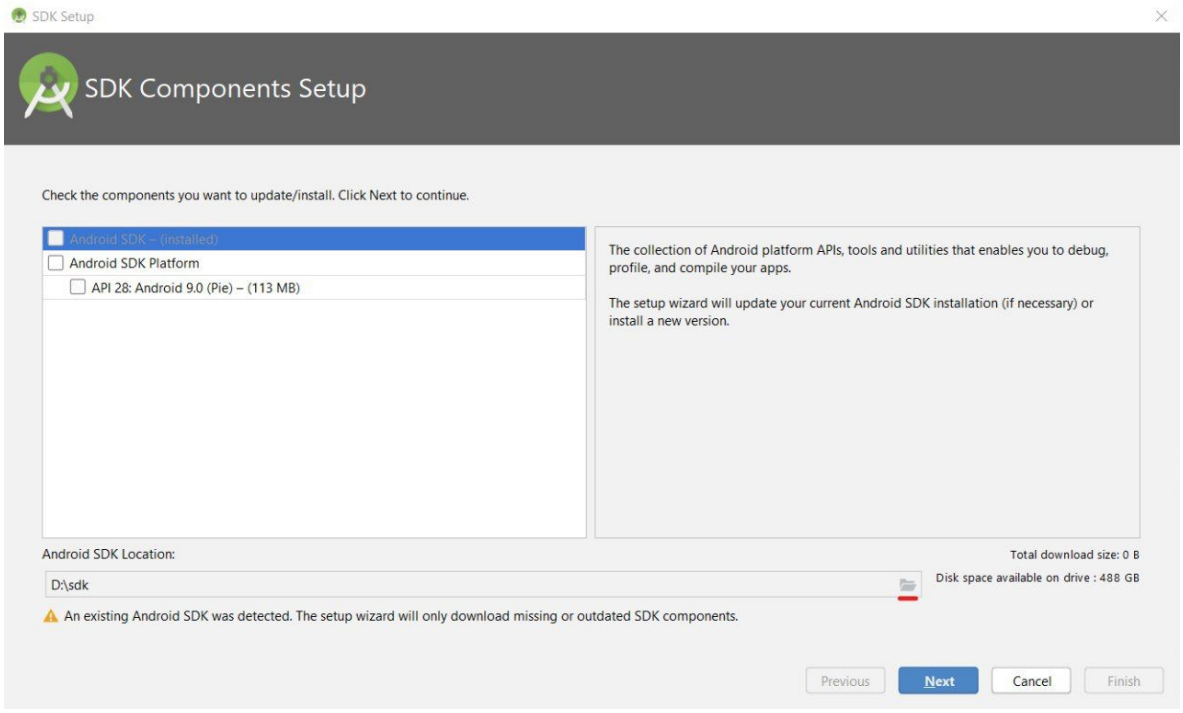
14. Common Problems (Windows)

A. *SDK package(s) is missing*

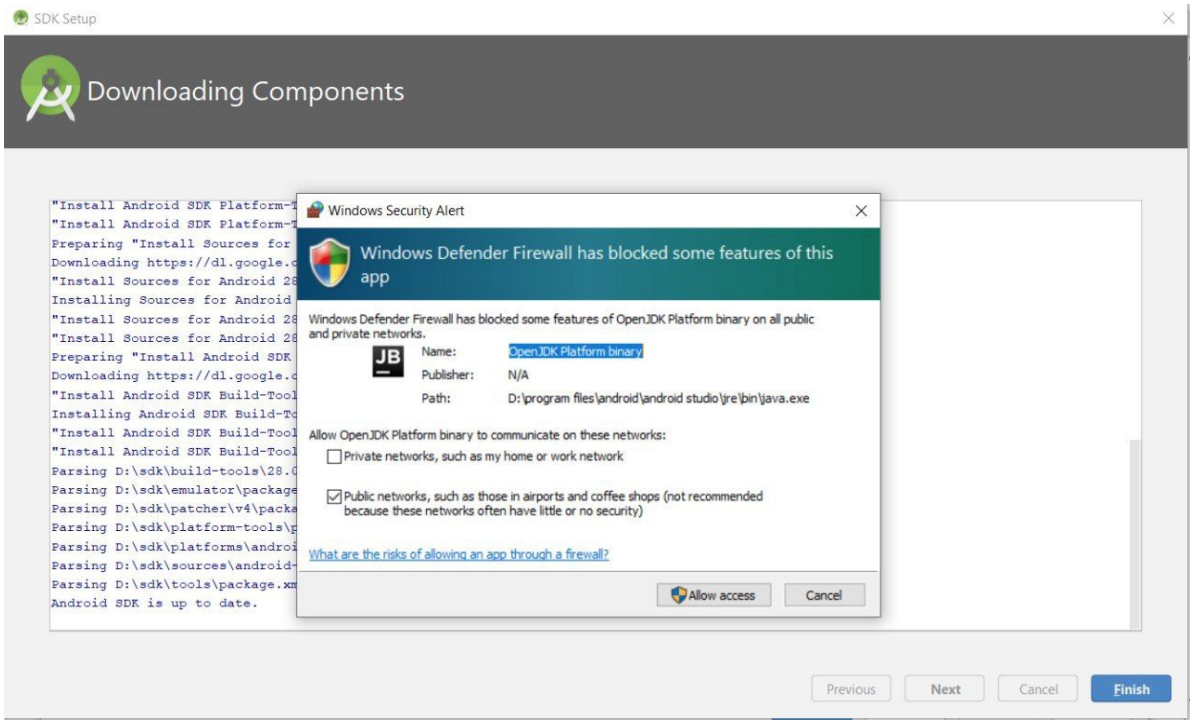
If the SDK package(s) is missing, go into **Tools > SDK Manager.** Once you open the SDA Manager, this window should pop-up. If you want to specify a path for Android SDK Location, you can click “Edit” and change it.



Click “Edit”, then click on the folder icon to specify a path for it. Once you are done, click next;



You may receive this message from Windows Firewall when it is done. For the purposes of this app, it is alright to select ‘Public networks’ and click ‘Allow Access’.

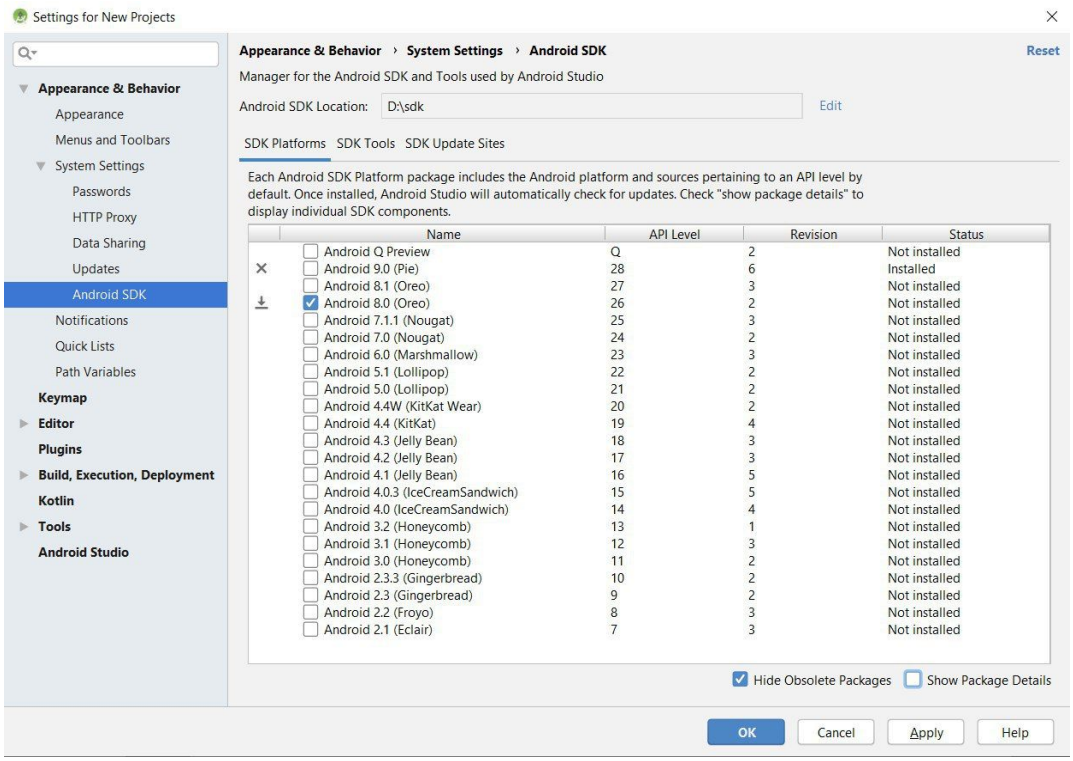


Once it’s done, the SDA package(s) should be installed.

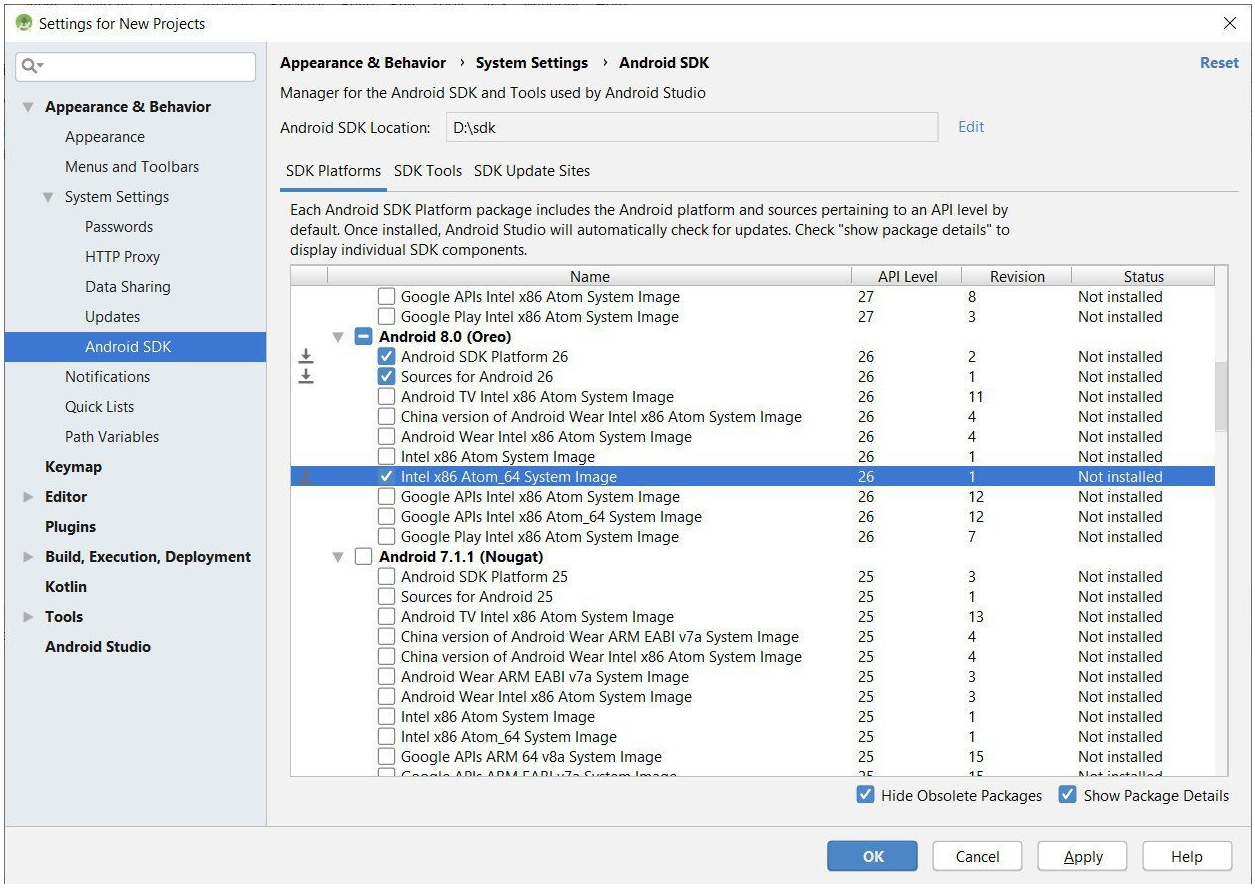
B. Emulator won’t show up

If you are running into a problem that your emulator won’t show up, try to complete the following steps:

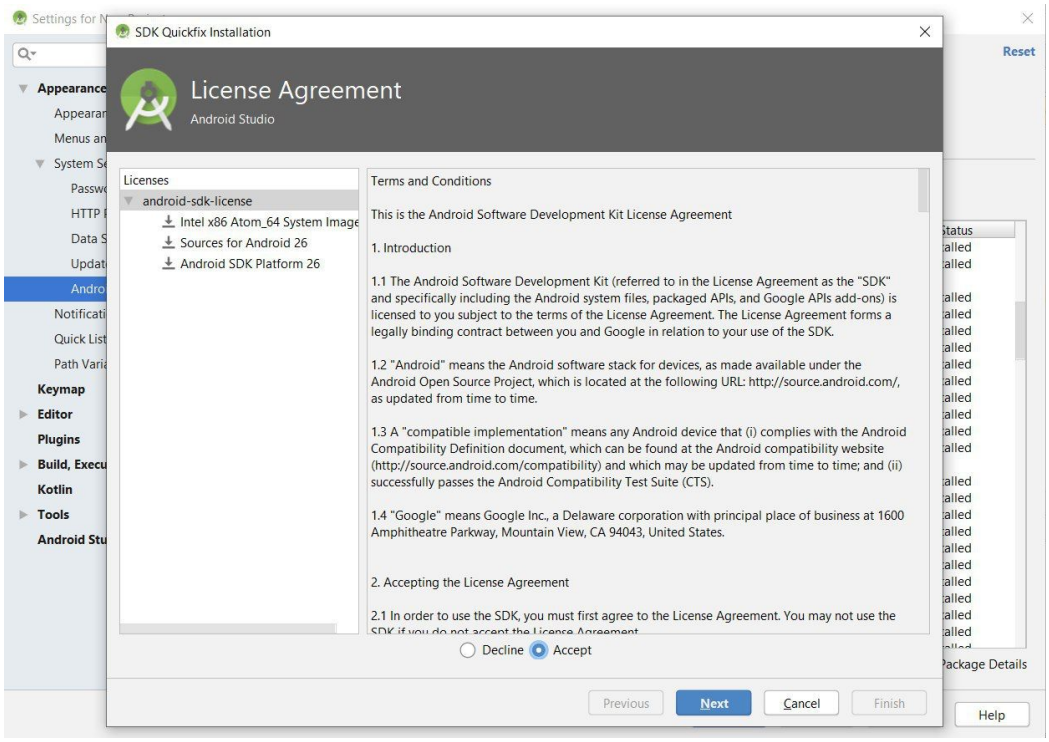
- Click **Tool -> SDK Manager**, then click on “Show Package Details”:



Depending on the version of your Windows Processor, (eg. 32-bit vs 64-bit), click on either **Intel x86 Atom System Image** for 32-bit or **Intel x86 Atom_64 System Image** for 64-bit. In our case, we have the 64-bit so we are going with the **Atom_64** one. Click Apply.



It will prompt you to accept the license Agreement, accept it and click next:



Once it's done, the SDA package(s) should be installed and emulator should be up and running.