**Android Studio for 1.1.2: Media Library**

Part V: Android Studio

In this part of the activity, you will develop an app to track favorite songs, books, and movies. You will use the Java code that you wrote in BlueJ to get started building the app in Android Studio.

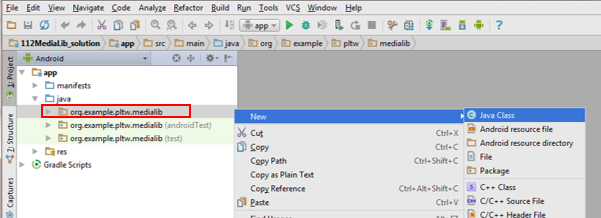
1. If you have not opened Android Studio before, refer to Activity 1.1.1 Introduction to Android Development and complete Part III.
2. Create a MediaLib folder in your AndroidProjects folder.
3. Get a copy of the 1.1.2MediaLibApp source files from your teacher. Copy or extract the files to a MediaLib folder in your AndroidProjects folder.
4. In Android Studio, import the MediaLib project: Select File > New > Import Project…
5. A dialog appears showing your file structure. Navigate to your AndroidProjects folder and then navigate to the location where you copied or extracted the MediaLib files. In the MediaLibfile structure, select a file named build.gradle. It will be in the MediaLibfolder, not in a subfolder. Click OK.
6. Use the Project panel to open MainActivity.java. This file is very similar to the main activity in HelloWorld except that in place of the sayHello method, you now have showMedia.
7. Run the app to confirm it is working. When you touch or click the **SHOW LIBRARY** button, you should see “none” displayed in the app.

In MainActivity, the showMedia method will perform the tasks that your main method did in your BlueJ application.

1. In BlueJ, copy the code that creates a song object and sets its title. Paste it into the showMedia method, either before or after the message that displays “none”.

Notice how some of the text turns red. This indicates an error. Android Studio pre-compilesyour code as you type and indicates any errors it finds. In this case, Song and all of its methods are undefined. You will now correct the errors.

1. In the project panel, right-click on **org.example.pltw.medialib**.

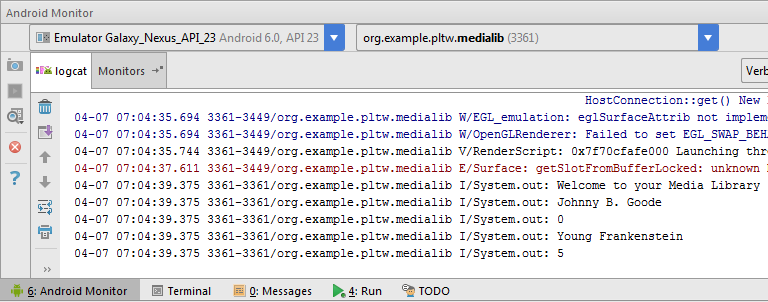


1. In the context menu that opens, select **New** > **Java Class**.
2. As you did in BlueJ, enter **Song** for the class name and click **OK**.
3. Copy the contents of your Song class in BlueJ to your Song class in Android Studio.

Your constructors and methods will be gray and underlined. This is just a warning indicating that the constructors and methods have not yet been used. You can ignore the warning for now.

1. Return to MainActivity. All of the errors that were in red should be gone.
2. Repeat the process to create Movie and Book classes. Be sure to include code in showMedia that creates a Movie and a Book object and sets and gets the titles.
3. Run your app and click the SHOW LIBRARY button.

Nothing has changed in the app on your device, but there should be output in a panel at the bottom of the IDE. This area is called the “logcat” (short for *log* and con*cat*enation).



1. Observe your own logcat panel to see the output from your System.out.println(…) lines of code.

Part VI: Work with TextView

The titles of your songs, movies, and books need to show on the app’s display.

1. Find the line of code in your program that reads:

|  |  |
| --- | --- |
| 1 | TextView outputText = (TextView) findViewById(R.id.*mediaLibText*); |

Later, you will learn what each part of this code does. For now, just know that this creates an outputText object that can show text on the app’s screen.

1. The next line uses your outputText object and calls the method setText.

|  |  |
| --- | --- |
| 1 | outputText.setText("none"); |

The setText method sets the text in the outputText object. The outputText object displays that text on the app’s display. Use code similar to this to display your song’s title in the app instead of printing it with System.out.println.

1. Delete the line of code that displays “(none)” and test your app.
2. In a similar way, display the title of your movie. Was the output what you expected? Why do you think only the movie title appears and not the song *and* the movie title?

You will now learn a powerful tool that is built into the Android Studio IDE. Your app needs to add text instead of setting text. You may guess that a method to add text might be called something like “addText”.

1. Backspace over the setText method name and type the letter a.

A context menu appears, showing all of the methods that are available for this object.

1. Try to guess how to add text to this object. The method is actually called append. Use it to display the movie’s title. Test your app.

Both titles should appear, but the output is not easy to read, because the titles run together. A blank line would help your output. In Java, to get a blank line, you need to use some special characters.

1. Add the following code after your setText method call, but before your append method call:

|  |  |
| --- | --- |
| 1 | outputText.append("\n"); |

The \n is an escape sequence that represents a new line.

1. Now to improve the output even more. Place a line of text that says “SONGS:” before your song title, a line of text that says “MOVIES:” before your movie title, and a line of text that says “BOOKS:” before your book title.
2. Create at least one more song, movie, and book, and show their titles. You need to come up with some new variable names other than song1, movie1.

Part VII: Encapsulation (and How to Violate It!)

Object-oriented languages use encapsulation, the process of defining data and their related methods together, in an object. Your instance fields, accessors, and mutators encapsulate the data for your songs, movies, and books. In the Android operating system, however, calling an accessor or mutator frequently can waste valuable resources and slow down your app. For this reason, Android apps sometimes break encapsulation. In this part of the activity, you will create a new public class that breaks encapsulation intentionally.

1. Create a new class called Greeter whose content is a *public* instance field instead of a private one:

|  |  |
| --- | --- |
| 1  2  3 | public class Greeter {  public String message = "Welcome to your Media Library"; } |

1. In the onCreate method of MainActivity, add the line to create a Greeter object and set the text using a TextView object named welcomeText.

|  |  |
| --- | --- |
| 1  2 | Greeter greeter = new Greeter();  welcomeText.setText(greeter.message); |

Notice how this breaks the encapsulation (the message instance field can be directly accessed without a method). You should never break encapsulation *without good cause*.

**Conclusion**

1. What are the advantages and disadvantages of using accessors and mutators in an Android app?