ACTIVITY **1.1.2**

Your First Class

INTRODUCTION

When you program in an object-oriented language, you are programming with objects. An **object** describes a set of data in a program, and it can represent anything—a text message, a list of contacts, a bicycle—anything that needs to be represented. In this activity, you will write a simple Media Library application. You will create objects that represent your favorite songs, movies, and books.

object
A representation of data that serves a similar purpose or is related in some way.

Materials

- Computer with BlueJ and Android™ Studio
- Android™ tablet and USB cable, or a device emulator

RESOURCES

Lesson 1.1 Reference Card for Basic Constructs
Resources available online

Procedure

Part I: Create a Song Class

1. As you did in the last *Activity 1.1.1 Intro to Android Development*, create a new project in BlueJ. Call the project **MediaLib** and create a new class called `MediaLib`.
2. As you did in your HelloWorld project, delete the code that BlueJ auto-generated, and replace it with a `main` similar to the `main` in HelloWorld. (To save time, use copy and paste.)
3. Change the message that prints to "`Welcome to your Media Library`".

   Your Media Library will eventually contain lists of your favorite media items, such as names of songs, movies, and books. Each of these categories will be a class in your program.
4. Learn more about **Classes and Objects** in the online Java Review material

5 Observe your `MediaLib` class in the BlueJ editor. Read **FirstClass and SecondClass** up through the section titled **Running a Java Program**. Compare your `MediaLib` class with `SecondClass`. What do they have in common?

6 In BlueJ, click **New Class...** and enter the name **Song**. By convention, classes have names that begin with an uppercase letter, so be sure to capitalize properly. Click **Ok**.

This creates a new file called `Song.java`, which defines your new class. The file is automatically included in your MediaLib project.

7 Edit your `Song` class (double-click its box or select **Open Editor** from the context menu) and take a closer look at the code that BlueJ generated. After the line with `public class Song` that begins the class definition, you will see:

```
// instance variables - replace the example below with your own
private int x;
```

This indicates a **variable**. **Learn about variables in Java**. A **primitive** data type is a built-in part of Java language, without using external definitions such as your `Song` object.

What are four variable types, in both the primitive and object categories?

The **int** variable x has been defined in your Song class and means that in addition to being an integer variable, it is also an **instance field** for your class. Instance fields contain the data for an object.

variable	A place in memory that stores a value.
primitive	A type of variable built into the Java language; for example, int, double, and boolean.
int	A type of primitive variable that can hold whole numbers such as 3, –76, 20393.
instance field	A variable in a class, used to describe data for the class, for example the title of a Song.

8 Read the section in the online resource titled **Fields - Instance Variables**.

What are the instance fields for the Name class?	Two strings: name and cell

9 You will define two instance fields to represent the data for your `Song` class. The first is a rating on a scale of 1 to 10. Change the name of variable `x` to `rating`.

10 Create a second data item for your Song class—the `title` of the song. This should be a **String** data type and also be `private`. You will learn much more about strings in later activities; for now, you only need to know they are just a sequence or "string" of characters.

String
An object in Java made up of a sequence or string of characters.

After the instance fields, the next few lines in your `Song` class are:

```
/**
 * Constructor for objects of class Song
 */
public Song()
{
     // initialise instance variables
     x = 0;
}
```

This part of your class, specifically lines 4–8 above, is called the **constructor**.

constructor
The code in a class definition that initializes the instance fields.

11 **Read about constructors**.

What is the role of a constructor?	A constructor initializes the instance fields of the class.

When you **initialize** a variable, you are assigning it a value for the first time.

initialize
Assign a value to a variable for the first time.

empty string
A string value that has no characters, represented in code as “”.

12 In the constructor, change the line of code that initializes `x` to initialize `rating` instead.

13 Add another line of code to your constructor so that the instance field `title` is initialized to a string without any characters:

```
title = "";
```

Notice there is no space between the quotation marks. This is called the **empty string** because it is a string without any characters.

14 Finally, delete the rest of the auto-generated code, but be sure to leave the last curly brace that closes or finishes your class definition.

15 Compile your code and fix any bugs you may have.

Part II: Make a Song Object

Creating a new object creates a place in memory where information about the object is stored. In this part of the activity, you will learn how to create a new object and how to access the information associated with the object.

16 Continuing in `MediaLib`, create a `Song` using the `new` keyword to **instantiate**, or create, a new object from the `Song` class:

```
Song song1 = new Song();
```

instantiate
To create an object from a class definition, for example new Song(“Twist and Shout”).

object reference
An object variable whose value points to a location in memory where the data for the object is stored.

The `song1` variable is an **object reference**. An object reference refers to an object in memory; it points to a place in memory where the data for an object is stored. In this case, `song1` points to where `title` and `rating` are stored as a `Song` object.

17 Use the following to print out the `song1` variable:

```
System.out.println(song1);
```

What is shown?

When you use `println` to show the contents of the `song1` variable, it shows the data type (`Song`) the @ sign, and then the reference or the address in memory of that object.

To show the actual data that the `song1` variable points to is called de-referencing an object. You will **de-reference** an object in the next section.

de-reference
Given an object reference, de-reference means retrieving the data for the object.

Part III: Create Methods

De-referencing an object means to access the information, or data, contained in the object. A common way to de-reference an object and *access* its data is to use an object **method** called an **accessor**. A common way to de-reference an object and *set or change* its data is to use a method called a **mutator**. In this part of the activity, you will discover how to get and set data in an object.

method
Similar to a function or procedure, a method is a collection of code that describes what an object can do, for example setTitle("").

18 Read the section titled **Methods**.

What could one accessor for your song class do?	One accessor for the Song class could get the title for get the rating

19 Write an accessor method for your `Song` class called `getTitle()`.

```
public String getTitle() {
     return title;
}
```

a. The above method is `public`, meaning other objects, including your `MediaLib` object can use or access this method. If you had written `private`, other classes would not be able to access this method.

b. After public, you specified `String`. This means that the `getTitle()` method will **return** or pass back some `String` value. Not all methods return something.

c. Then there is the contents or the body of your method. Here, the method simply returns the title of your song.

return or return value

The value that a method passes back to the caller.

20 Now write a mutator method for your `Song` class called `setTitle()`.

```
public void setTitle(String t) {
    title = t;
}
```

a. The above method is also `public`, meaning other objects, including your `MediaLib` object can use or access this method.

b. After public, you specified `void`. This method returns nothing. Notice it does not have a `return` statement like your accessor does.

c. After the method name `setTitle`, you specified `(String t)`. This means a string value is passed to your method and assigned to the variable `t`. Think of `t` as the input to your method.

d. Finally, you assigned the value of `t` to `title`.

21 With accessor and mutator methods defined, you can get and set the title of a song. In the `main` method of `MediaLib`, add:

```
song1.setTitle("Johnny B. Goode");
System.out.println(song1.getTitle());
```

22 In the same way you just created an accessor and a mutator for your `title` instance field, create an accessor and a mutator for the `rating` instance field. Show the rating of your song using `System.out.println(…)`. Remember that rating is an `int` and not a `String`.

This may seem like a lot of work just to manage a bit of data, but it keeps data secure and consistent. Suppose, for example, you want to keep all ratings between 1 and 10, or between 1 and 5. Your mutator can make sure that happens.

Part IV: Create Movies and Books

Practice what you have learned by creating two new classes to keep track of movies and books. Refer to the code that is already written for the song class as an example.

23 In the same way you created a `Song` class, the instance fields, and the mutators and accessors, create a `Movie` class with the same instance fields, mutators, and accessors.

24 Repeat the process to create a `Book` class with the same instance fields and mutators and accessors.

25 Create a `Movie` and a `Book` object in the `main` method of `MediaLib` and show the data with: `System.out.println(…);`

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.

26 If you have not opened Android Studio before, refer to *Activity 1.1.1 Introduction to Android Development* and complete Part III.

27 Create a *`MediaLib`* folder in your *`AndroidProjects`* folder.

28 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.

29 In Android Studio, import the MediaLib project: Select **File** > **New** > **Import Project…**

30 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 *`MediaLib`* file structure, select a file named *`build.gradle`*. It will be in the *`MediaLib`* folder, not in a subfolder. Click **OK**.

31 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`.

32 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.

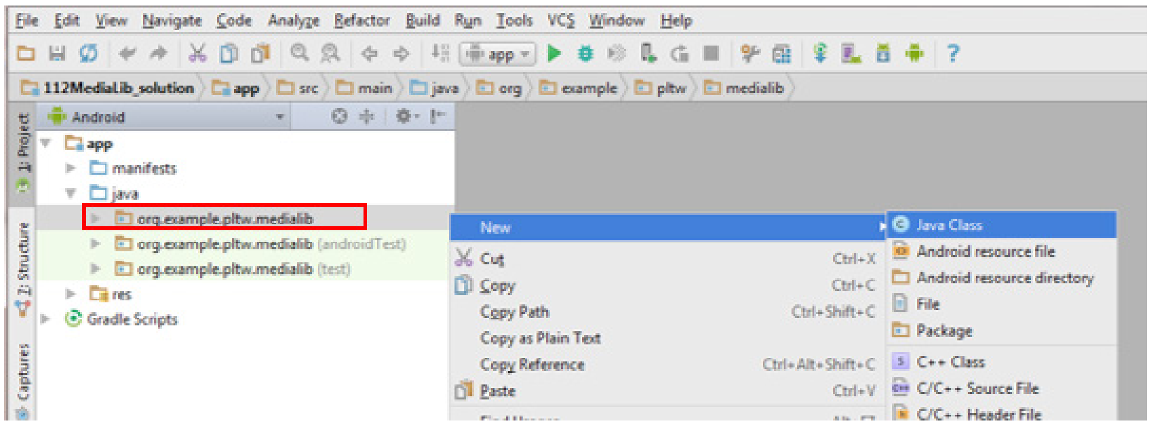
In `MainActivity`, the `showMedia` method will perform the tasks that your `main` method did in your BlueJ application.

33 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”.

NOTE

Notice how some of the text turns red. This indicates an error. Android Studio pre-compiles your 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.

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



File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
112MediaLib_solution app src main java org example pltw medialib
Android
app
manifests
java
org.example.pltw.medialib
org.example.pltw.medialib (androidTest)
org.example.pltw.medialib (test)
res
Gradle Scripts
1: Project
2: Structure
Captures
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Java Class
Android resource file
Android resource directory
File
Package
C++ Class
C/C++ Source File
C/C++ Header File

35 In the context menu that opens, select **New** > **Java Class**.

36 As you did in BlueJ, enter **Song** for the class name and click **OK**.

37 Copy the contents of your `Song` class in BlueJ to your `Song` class in Android Studio.

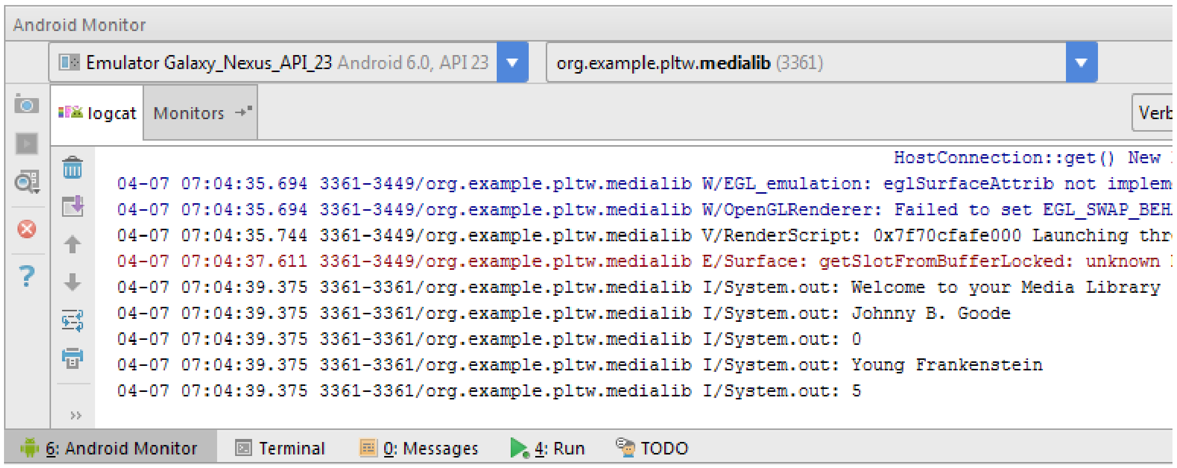
Your constructors and methods will be gray and underlined. This is just a warning indicating that the constructors and methods have not yet been used. You can ignore the warning for now.

38 Return to `MainActivity`. All of the errors that were in red should be gone.

39 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.

40 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 concat*enation*).



Android Monitor
Emulator Galaxy_Nexus_API_23 Android 6.0, API 23
org.example.pltw.medialib (3361)
logcat
Monitors
Verb
6: Android Monitor
Terminal
0: Messages
4: Run
TODO

```
                                                                HostConnection::get() New
04-07 07:04:35.694 3361-3449/org.example.pltw.medialib W/EGL_emulation: eglSurfaceAttrib not implem
04-07 07:04:35.694 3361-3449/org.example.pltw.medialib W/OpenGLRenderer: Failed to set EGL_SWAP_BEH
04-07 07:04:35.744 3361-3449/org.example.pltw.medialib V/RenderScript: 0x7f70cfafe000 Launching thr
04-07 07:04:37.611 3361-3449/org.example.pltw.medialib E/Surface: getSlotFromBufferLocked: unknown
04-07 07:04:39.375 3361-3361/org.example.pltw.medialib I/System.out: Welcome to your Media Library
04-07 07:04:39.375 3361-3361/org.example.pltw.medialib I/System.out: Johnny B. Goode
04-07 07:04:39.375 3361-3361/org.example.pltw.medialib I/System.out: 0
04-07 07:04:39.375 3361-3361/org.example.pltw.medialib I/System.out: Young Frankenstein
04-07 07:04:39.375 3361-3361/org.example.pltw.medialib I/System.out: 5
```

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

Part VI: Work with Text View

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

42 Find the line of code in your program that reads:

```
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.

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

```
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`.

44 Delete the line of code that displays "(none)" and test your app.

45 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".

46 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.

47 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.

48 Add the following code after your `setText` method call, but before your `append` method call:

```
outputText.append("\n");
```

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

escape sequence
A sequence of characters that have special meaning in Java; for example, \n represents a newline.

49 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.

50 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.

encapsulation
The process of keeping related data and methods together and defining an object for them.

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

```
public class Greeter {
    public String message = "Welcome to your Media Library";
}
```

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

```
Greeter greeter = new Greeter();
welcomeText.setText(greeter.message);
```

NOTE

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?
2. There is a lot of duplication of code between your song, movie, and book classes. What instance fields and methods do they all have? Can you think of a way to reduce this duplication?