**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.

**Materials**

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

Activity

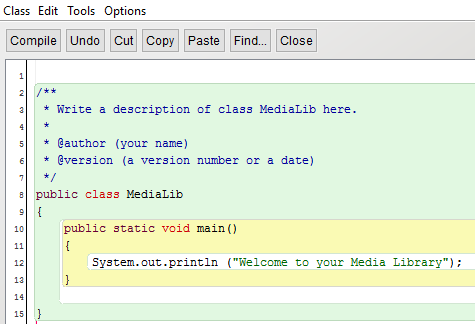
Part I: Create a Song **Class**

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.

**Project > New Project… > (Navigate to your OneDrive >**

**BlueJProjects folder) > Type name of project**

1. 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.)



1. 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.

1. Learn more about [**Classes and Objects**](http://interactivepython.org/runestone/static/JavaReview/JavaBasics/introClassObject.html) in the online Java Review material; continue until you get to **Intro to Java - Summary**. If the activities don’t work in your favorite browser, try copying and pasting the address in a different browser (Firefox, Chrome, IE). Read the review and answer the 3 questions at the bottom. Paste a screenshot of your answers between questions #4 and 5 of **What is a Class and an Object?**.



1. Observe your MediaLib class in the BlueJ editor. Read [**FirstClass and SecondClass**](http://interactivepython.org/runestone/static/JavaReview/JavaBasics/firstClass.html) up through the section titled **Running a Java Program**. Compare your MediaLib class with SecondClass. What do they have in common?

The two classes are very similar, with the only differences being a different class name in the class declaration and a different String argument for the println call.

1. 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.

1. 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:

|  |  |
| --- | --- |
| 1  2 | // instance variables - replace the example below with your own  private int x; |

This indicates a variable. [**Learn about variables in Java**](http://interactivepython.org/runestone/static/JavaReview/VariableBasics/vIntro.html). 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?

int

double

boolean

String

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.

1. Read the section in the online resource titled [**Fields - Instance Variables**](http://interactivepython.org/runestone/static/JavaReview/JavaBasics/firstClass.html#fields-instance-variables). What are the instance fields for the Person class?

private String name;

private String email;

private String phoneNumber;

// Person class found at <http://interactivepython.org/runestone/static/JavaReview/OOBasics/objects&Classes.html?highlight=class%20person>

1. 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.
2. 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.

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | /\*\*  \* 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.

1. [**Read about constructors**](http://interactivepython.org/runestone/static/JavaReview/JavaBasics/firstClass.html#constructors). What is the role of a constructor?

Constructors set the values for the fields when the class is first instantiated.

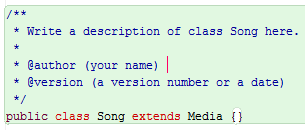
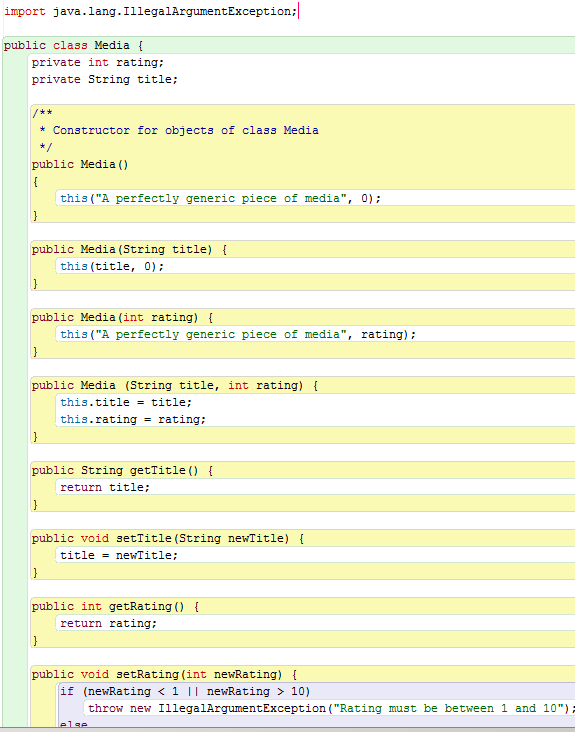
When you initialize a variable, you are assigning it a value for the first time. You may have noticed that your class no longer has an x instance field, but it does have a rating instance field.

1. In the constructor, change the line of code that initializes x to initialize rating instead.
2. Add another line of code to your constructor so that the instance field title is initialized to a string without any characters:

|  |  |
| --- | --- |
| 1 | title = ""; |

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

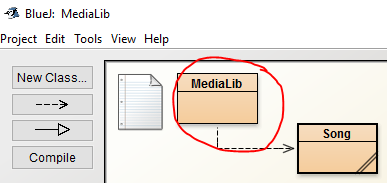
1. Finally, delete the rest of the auto-generated code, but be sure to leave the last curly brace that closes or finishes your class definition.
2. Compile your code and fix any bugs you may have. Paste a snapshot of your Song class in the space below.



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.

1. Continuing in MediaLib, create a Song using the new keyword to instantiate, or create, a new object from the Song class:



Right click on MediaLib > Open Editor

|  |  |
| --- | --- |
| 1 | Song song1 = new Song(); |

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.

1. Use the following to print out the song1 variable:

|  |  |
| --- | --- |
| 1 | System.out.println(song1); |

What is shown?

Welcome to your Media Library

Song@47f931de

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-referencingan object*.* You will de-reference an object in the next section.

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.

1. Read the section titled [**Methods**](http://interactivepython.org/runestone/static/JavaReview/JavaBasics/firstClass.html#methods). What could one accessor for your Song class do?

Get the title of the song.

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

|  |  |
| --- | --- |
| 1  2  3 | public String getTitle() {  return title;  } |

* 1. 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.
  2. After public, you specified String. This means that the getTitle() method will return or pass back some String value. Not all methods return something.
  3. Then there is the contents or the body of your method. Here, the method simply returns the title of your song.

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

|  |  |
| --- | --- |
| 1  2  3 | public void setTitle(String t) {  title = t;  } |

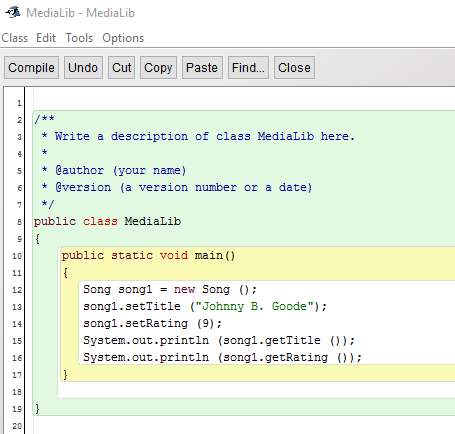
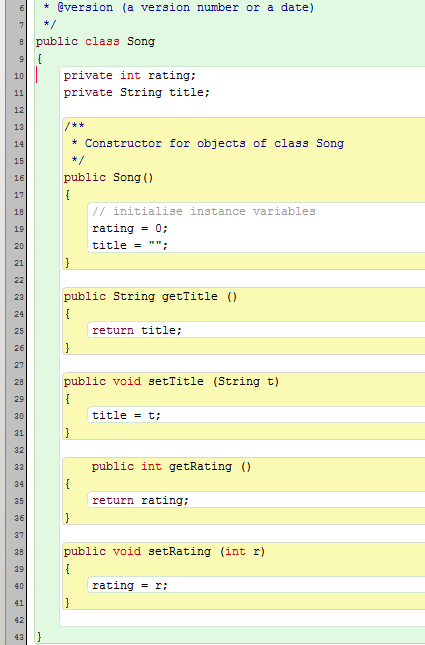
1. The above method is also public, meaning other objects, including your MediaLib object can use or access this method.
2. After public, you specified void. This method returns nothing. Notice it does not have a return statement like your accessor does.
3. After the method name setTitle, you specified (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.
4. Finally, you assigned the value of t to title.
5. With accessor and mutator methods defined, you can get and set the title of a song. In the main method of MediaLib, add:

|  |  |
| --- | --- |
| 1  2 | song1.setTitle("Johnny B. Goode");  System.out.println(song1.getTitle()); |

1. 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.

Screenshot of MediaLib class Screenshot of Song class

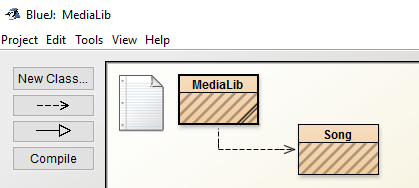


Talk to us if you are having difficulty with the code.

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.

1. 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.



1. Repeat the process to create a Book class with the same instance fields and mutators and accessors.
2. Create a Movie and a Book object in the main method of MediaLib and show the data with: System.out.println(…);
3. Create a zip folder for your MediaLib files and submit on PowerSchool.

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.

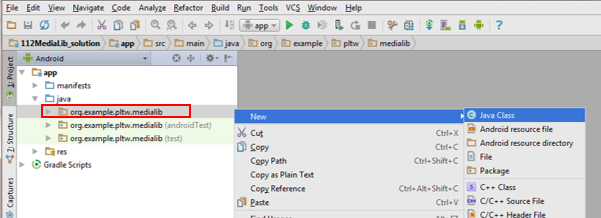
1. If you have not opened Android Studio before, refer to Activity 1.1.1 Introduction to Android Development and complete Part III.
2. Create a MediaLib folder in your AndroidProjects folder.
3. Get a copy of the 1.1.2MediaLibApp source files from your teacher. Copy or extract the files to a MediaLib 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**.

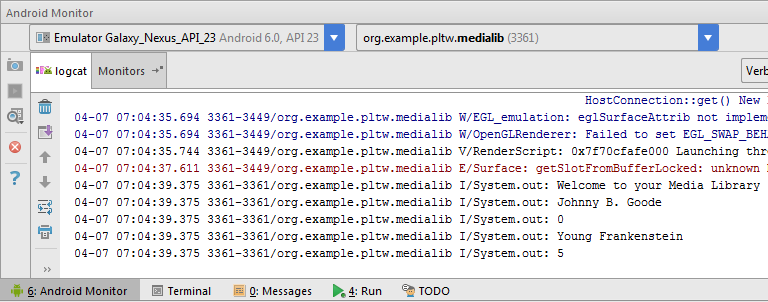


1. In the context menu that opens, select **New** > **Java Class**.
2. As you did in BlueJ, enter **Song** for the class name and click **OK**.
3. Copy the contents of your Song class in BlueJ to your Song class in Android Studio.

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?
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?