ACTIVITY **1.1.3**

Making Objects

INTRODUCTION

In this activity, you will create many `Song` objects and add a new feature to your MediaLib project to store the cost of media items. Your app will be able to add up total cost, determine the average cost, and even apply discounts to some of your items. To do all of this, you will need some more knowledge about variables and different data types in Java.

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

Free Response Question: Book
Resources available online

Procedure

Part I: One Class, Many Constructors

You have already created the Song class, but now you need to create many song objects. In this part of the activity, you will learn how to create constructors that makes it easier to instantiate objects in a class

1. Open your MediaLib project in BlueJ, and open the `Song` class in the editor.
2. When you created the `title` and `rating` instance fields, **you *declared* them**. Using variables from your media library project, show an example of assignment dyslexia as described in the online resource.
3. Declare a new instance field called `price` in your `Song` class. Initialize it to `0.0` in the `Song` constructor. Check to be sure you made this a `private` instance field.

4. Provide an accessor and a mutator method for `price`.
   a. The accessor must return a data type that is the same as price, that is, a `double`.
   b. The mutator must also specify a `double` data type for its parameter.
5. In the `main` method (in your `MediaLib` class), use your newly written mutator and accessor methods to:
   a. Show the price of a song in the console.
   b. Assign a `price` for a song.
6. Back in your `Song` class, declare a `boolean` instance field called `favorite`.

   Not all accessors and mutators need to use “get” or “set” in the method name. This is especially true when getting and setting `boolean` values.
7. Use the following mutator as a mutator of the `favorite` variable.

```
public void addToFavorites() {
    favorite = true;
}
```

8. If you wanted to write one mutator method that could set the `favorite` variable to any `boolean` value, what might that method look like? Which version of the mutator for `favorite` do you prefer?

Eventually, you will create a lot of songs. With your current code, it may seem tedious to create many songs and set all of the titles, ratings, and prices. An easier way to create them would be to use one constructor and no accessors or mutators. It is possible to do this with your code—you just need a new constructor.

In general, a class can have more than one constructor as long as those constructors are different from one another. For example, the following is how you might create or instantiate a `Song` object using a constructor that has a title and a price:

```
Song song1 = new Song("Respect", 1.29);
```

The constructor would look like:

```
public Song(String title, double price) {
    this.title = title;
    this.price = price;
}
```

The declaration for the constructor, `public Song(String title, double price)`, has two variables passed to it. There is a special name for this type of variable—**parameter**. The first parameter is `title` and the second parameter is `price`.

parameter
A variable that is passed into a method.

In the body of the constructor, the `this.title` syntax refers to the instance field `title`. With the `this.title = title` syntax, you are setting the `title` *instance field* to the value that was passed in using the `title` *parameter*. Using the same name for parameters and instance fields is a common practice in Java constructors. The `this.` syntax identifies which `title` variable to reference.

9 Copy and paste the code for the new constructor into your `Song` class. Compile and fix any bugs.

10 Using the new constructor as a template, create a third `Song` constructor that has a rating as its third parameter. Be sure to assign `this.rating` to the new parameter in this third constructor. Compile and check for bugs.

11 Using the constructor that has three parameters, create new songs so that you have at least seven or eight songs in your library. The titles don't really matter at this point, but use a price of 1.29 or .99 for different songs and vary the ratings.

Part II: Variables and Calculations

Java uses the standard mathematical operators to do calculations:

+	Addition
-	Subtraction
*	Multiplication
/	Division

You will use these mathematical operators to do some calculations in this section.

12 First, calculate the total cost of all your songs. To do this, declare an accumulator variable. Name this `double` variable `totalCost`. Notice the unique way this variable is named.
Learn about naming variables in the Java language.

a. What is this naming convention called where each word, or each word after the first, starts with an uppercase letter?

b. Why should `totalCost` have a data type of `double`?

13 In addition to total cost, keep track of the number of songs you have. Create a variable called `numSongs`.

What data type do you think this variable should be? Compile and check for bugs.

14 **Learn how to change variables in Java**. Every time you create a song, add the price to `totalCost`. Then add 1 to `numSongs`.

15 In a similar way, create a `totalRatings` variable and add all ratings of songs to it. Compile and check for bugs.

Part III: Calculation Errors

You have the total cost of all songs and the total number of songs. In this part of the activity, create a new way to find out the average cost of a song in your media library.

16 You can now calculate the average cost:

a. Create a well-named variable to store the average cost.

b. Divide the total cost by the number of songs you have and store the result in your new variable.

c. Using `System.out.println(…)`, display the total cost and the average cost of all songs.

17 The calculated cost may not be what you expected. For example, add this code fragment somewhere in your main method:

```java
// testing a calculation:
double testVal = 109.41;
double testResult = testVal / 10;
System.out.println("Testing Result:");
System.out.println(testResult);
```

The result of this calculation is 10.940999999999999. This may seem strange, since 109.41 divided by 10 should be 10.941. Computers store all data in a binary language of 1's and 0's (which represents the state of electricity flowing or not flowing). Certain numbers in our decimal system cannot be represented accurately in the binary number system on a computer. Doing calculations with decimals causes round-off errors. For now, you can ignore this round-off error.

18 In the same way you calculated an average cost, calculate an average rating for all of your songs and display it. Check the calculation manually to see whether your average is correct. It may not be!

19 In addition to round-off errors, calculations in Java can be incorrect due to the data type of the variables. Division using integers or doubles can change the results. **Learn how *casting* variables can solve this problem**.

20 To correctly calculate the average, the variable that stores the average rating (for example, `aveRating`) should be a `double`, `totalRating` should be an `int`, and `numSongs` should be an `int`.

21 Use a *cast* to get the correct result and check your calculations manually.

Part IV: The Modulus Operator

In addition to the standard math operators +, -, *, and /, Java has a modulus operator represented by the % symbol. In this part of the activity, you will learn how to use the modulus operator to convert number of minutes into hours and minutes.

22 **Learn how to use the modulus operator**. For now, skip the discussion of the `Math` class and its random method, but be sure to do the practice questions using the modulus operator.

23 Create an integer `duration` variable for your `Movie` class and mutator and accessor methods for it.

24 Create a new method for your `Movie` class that displays the duration of the movie in a user-friendly format in hours and minutes.

25 Test your new method using durations such as 97, 134, or 199.

26 Improve the output: Display all output on one line using another method of the `System.out` class called `print(…)` that does not print a new line. Use syntax similar to:

```
System.out.print("Hours");
```

Part V: Android Studio

Transfer the code you have written so far in BlueJ into a new app in Android Studio.

27 If you have not opened Android Studio before, complete the following:

a. In *Activity 1.1.1 Introduction to Android Studio*; Part III.

b. In *Activity 1.1.2 Your First Class*; Parts V, VI, and VII.

28 Open your MediaLib project for Android Studio and make the changes to your `Song` class that you made in BlueJ.

29 Use your new constructors to create `Song` objects (copy/paste from BlueJ) in your `MainActivity showMedia` method. For now, don't show the output on your device; just make sure the new constructors and methods work by checking the output in logcat.

30 Make the same type of changes to your `Movie` and `Book` classes. Don't forget about the mutators and accessors for price.

31 To test whether all is well, create at least one Movie and one Book using the new constructors in `showMedia`.

This process of writing code in a simple IDE and inserting it into code in a more complex IDE is known as *prototyping*. Prototyping can be very useful when you want to test out some code before placing it in a more complex system.

Part VI: Android Studio Bugs

You have had some practice fixing bugs in BlueJ. Now, practice fixing bugs in Android Studio.

32 First, display line numbers in your editor window. Select **File** > **Settings...** In the search bar of the dialog that appears, enter **line numbers**. A Settings structure appears with **Editor > General** > **Appearance**. Click **Appearance**. Check the box to **Show line numbers** and then click **OK**.

33 Remove the semicolon at the end of a line of code anywhere in the `showMedia` method. What happens to your code?

34 Attempt to run your app. In the **Messages Gradle Build** panel at the bottom of the Android Studio window, you will see a message that states `error: ';' expected`.

35 Double-click the error message. Android Studio should place your cursor where the error occurs in your code. Fix the error.

36 Remove the last curly brace at the end of the file and compile. What error appears when you hover your mouse over the red squiggle? Fix the error.

37 Remove the closing double-quote in one of your strings. What error appears when you hover your mouse over the red squiggle? Fix the error.

38 Remove the first double-quote in one of your strings. Depending on what your string contains, you may get several different errors. What are the error(s) you see? As in BlueJ, the compiler tries to guess what you are trying to do in your code, and sometimes it guesses wrong. Fix the error.

39 Remove the period between `outputText` and `append`. What is the error that appears when you hover your mouse over the red text? Fix the error.

40 Remove the entire string in an `append` method call, so the line of text appears as `outputText.append();`.

a. What is the error? This message states that the compiler cannot find an `append()` method call that has no parameters.

b. On a new line of code, enter `outputText`. A context menu should appear. What are the two `append` methods you can use with this `outputText` object? (Notice that `CharSequence` is an Android-specific version of a `String` object and that you will only be using the first version of the `append` method in your apps.)

c. Use the "undo" feature in Android Studio, **<CTRL>z** to undo your errors. If you undo too many times, use **<CTRL><SHIFT>z** to "redo". These features are also available in the **Edit** menu.

41 Test your app to be sure you have fixed all errors.

42 Finally, you may notice "gray squiggles" throughout your code, especially in the `Song`, `Movie`, and `Book` classes. These are warnings, usually letting you know that a method or variable is unused. You can ignore them for now.

CONCLUSION

1. Why would you have more than one constructor for a class? Include sample constructors in your explanation.

AP Focus: Complete the **Free Response Question (FRQ): Book**.