Activity 1.1.3 Making Objects

Otakar Andrysek

Subtarget 1.2: Build a class with multiple constructors, each taking in a different number of arguments.

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~~

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.

   Done.

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.

   int apple;
   Correct: apple = 4;
   Incorrect: 4 = apple;

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.

   Done.

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.

      Done.

   b. The mutator must also specify a double data type for its parameter.

      Done.

5. In the main method (in your MediaLib class), use your newly written mutator and accessor methods to:
   a. Assign a price for a song.

      Done.

   b. Show the price of a song in the console.

      Done.

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.

Okay.

7. Use the following mutator as a mutator of the favorite variable.

```
1    public void addToFavorites() {
2        favorite = true;
3    }
```

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

   Where F is set in MediaLib to True/False.

   public void setFavorite(boolean favorite_mutator)
   {
   favorite = favorite_mutator;
   }

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:

```
1    public Song(String title, double price) {
2        this.title = title;
3        this.price = price;
4    }
```

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.

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.

   Done.

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.

    Done.

11. Using the constructor that has three parameters, create new songs so that you have at least ten 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.

    Done

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?

       CamelCase

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

       Because it might have a remainder (cents) and we do not want to lose that information.

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.

    Integer, it will always be a whole number.

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

    Done.

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

    Done.

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.

    Done.

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

```
1          // testing a calculation:
2          double testVal = 109.41;
3          double testResult  = testVal / 10;
4          System.out.println("Testing Result:");
5          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.

    Yay! 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!

    Done, but average is incorrect.

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**.

    Makes sense.

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.

    Got it.

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

    It worked!

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.

    Done.

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

    Done.

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.

    Done.

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

    Done.

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:

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

    Done.

Conclusion

1. Why would you have more than one constructor for a class?

   You can create a default object, but also create an object with user specified parameters. Allowing the coder or user to create object in different ways.

2. Paste snips below of your Song constructors:

```
/**
 * Constructor for objects of class Song
 */
public Song()
{
    // Initialise instance variables
    rating = 0;
    title = "";
    price = 0.00;
    favorite = false;
}
```

```
public Song(String title, double price, int rating) {
        this.title = title;
        this.price = price;
        this.rating = rating;
        this.totalCost += price;
        this.numSongs += 1;
        this.totalRatings += rating;
        this.averageCost = totalCost / numSongs;
}
```

3. Paste a snip below of your output after running your main method. It should show the following: The names of your songs, along with the price and rating and at the end of the list the average song price and rating; the names of your movies (there should be 3), along with the duration of the movie displayed in hours and minutes, the total duration of the 3 movies combined and the average movie length…the durations should be expressed as hours and minutes.

```
Title: Goodbye
Price: 1.29
Rating: 4
Title: Shoe
Price: 0.99
Rating: 8
Title: Water
Price: 0.99
Rating: 3
Title: Cow
Price: 1.29
Rating: 8
Title: Grass
Price: 0.99
Rating: 10
Title: Fire
Price: 0.99
Rating: 5
Title: Computer
Price: 0.99
Rating: 3
Title: Hamster
Price: 1.29
Rating: 6
Title: Oregon
Price: 0.99
Rating: 5
11.1
10
55
1.1099999999999999

Cars
8
This movie is 2 hours and 34 minutes.
Hobbit
4
This movie is 2 hours and 33 minutes.
Rabbit
7
This movie is 1 hours and 40 minutes.
Hours: 6 Minutes: 47
Hours: 2 Minutes: 15.0

Book@266a4705
Calculus
1
```

4. Study your classes…we may try next period to see how well you can make one from scratch.

All my code for this year will be available on GitHub!
https://github.com/otakar-sst/CSA/tree/master/Java%20Programs/Lesson%201