Revised Activity 1.2.1 Parsing Text

Otakar Andrysek

Subtargets:
1.5 Read text from and write text to data file so as to store persistent data
1.6 Use Integer class to convert strings to ints
1.7 Use for and while loops
I will look at 3, 8, 18, 19, 21, 30 and C1.

Introduction

Much of the data we use in the world, whether it's on your phone, on the web, or stored as other digital media, is stored as text. In this activity, you will write and read text to and from a file. With your data saved to a file on a hard drive (or flash drive), your data becomes more permanent, or **persistent**. Instead of disappearing every time you exit, your data will still be there the next time you run the program or app.

Materials

- Computer with BlueJ

Activity

Part I: Write Simple Text

In Part 1, you will begin by adding a new class called MediaFile to your MediaLib BlueJ project.

1. In BlueJ, open your MediaLib project from *Activity 1.1.3 Making Objects*.

   Done.

2. Get a copy of the BlueJ starter code *1.2.1MediaLib_StarterCode_BlueJ* from your teacher and copy or extract the file *MediaFile.java*. Drag-and-drop *MediaFile.java* into your MediaLib project window.

   Done.

The MediaFile class is responsible for reading and writing data to a file called *mymedia.txt*. The *mymedia.txt* file will be stored with your project and will contain your song titles and ratings.

3. Open *MediaFile.java*.

   Done.

   What are the methods of this class?

   readString( )
   writeString( )
   saveAndClose( )

   Be sure to include parameters and return types. Using your own words, write a brief description of each method.

   readString( ) - Reads a data file using a scanner, returns 'null' when 'done'.
   writeString( ) - Writes a string to a given file.
   saveAndClose( ) - Closes the file based on if the program is reading or writing.

4. In the main method of your MediaLib class, do the following:
   a. Use an accessor method to get the title of one of your songs.
   ```
   String t = song2.getTitle();
   ```
   b. Write the title to the output file.
   ```
   MediaFile.writeString(t);
   ```
   The above two lines of code can be combined into one step, which is a common, and often preferred, way to write this code. Combining code makes a method (and class) more compact and less "wordy" or verbose,

especially in Android development, where entire *classes* can be defined between the parentheses of the method.

c. Combine the two lines of code so they appear as:

```
MediaFile.writeString(song2.getTitle());
```

5. Save your data and close the file.

```
MediaFile.saveAndClose();
```

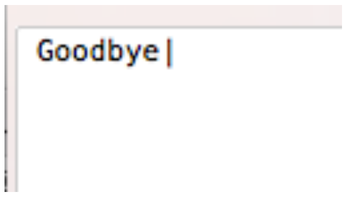
6. Run your program.
   Nothing appears to have changed because you did not change the data that was displayed. But, data should have been written to *mymedia.txt*.

   Indeed.

7. Navigate to this project's folder in your BlueJProjects folder.
   In addition to the java files and class files, you should see a file called *mymedia.txt*.

   I see!

8. Open this file using a program such as Notepad++ (Windows) or TextEdit (Mac). ~~Record what is in that file~~.Paste a snip of what you see in the file.



```
Goodbye
```

9. Close your *mymedia.txt* file.

   Done.

Part II: Write "Delimited" Text

Suppose you wanted to store a song's rating in addition to its title in the *mymedia.txt* data file. How do you think you could do this? A common way to store different data items in a single data file is to separate each data item with a **delimiting character**. This means the data is separated by a special character, usually one not found in data sets.

For example, the comma is a common delimiting character for numeric data. But for *mymedia.txt*, commas can occur in song titles and a different delimiting character is needed. The vertical bar, or **pipe** (|) is a good choice. A delimited line of text for a student and his or her student ID might look like:

Jane Doe|112233

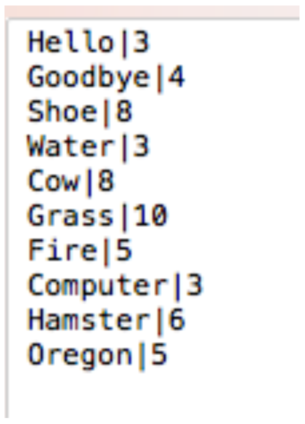
10. In MediaLib, use the writeString method and string concatenation to write the song title, a single vertical bar, and the rating of a song you created with your Song class. Run the program and confirm that your *mymedia.txt* file contains something similar to:
    The Twist|10

    Done.

11. After you confirm *mymedia.txt* is correct, be sure to close the file.

    Done.

12. In MediaLib, write all of your song titles and ratings to your data file using this syntax. Confirm that your data file contains clean and correct data.
    Paste a snip of what you see in the data file here.



```
Hello|3
Goodbye|4
Shoe|8
Water|3
Cow|8
Grass|10
Fire|5
Computer|3
Hamster|6
Oregon|5
```

Part III: Looping

At this point, you may have a sense of *duplicating code*, doing similar steps over and over. Programming languages have a construct that can help you eliminate duplication of code. The construct is called **iteration** or **looping**. In this part of the activity, you will write a loop to access all of the songs in your media library.

13. **Learn the different types of loops in Java**.
14. The MediaLib class is getting pretty crowded, so create a new class LoopingMediaLib with the following code:

```
public class LoopingMediaLib
{
    public static void main()
    {

    }
}
```

15. In the main method of LoopingMediaLib, use the MediaFile method readString to read from your *mymedia.txt* file.

```
        String songInfo = MediaFile.readString();
        System.out.println(songInfo);
```

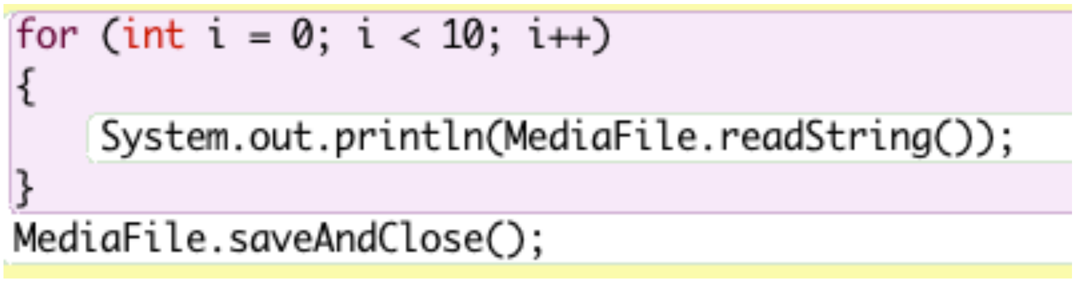
16. As you did before, combine these two lines of code into one. In addition to making your code less "wordy" or verbose, what else was eliminated when you combined these lines?

    An extra variable.

17. To read all of your songs from the data file without duplicating a lot of code, **learn the for loop**, and complete all of the activities.

    Done.

18. Write a for loop that reads and displays all of the songs defined in your data file.
    Paste a snip of the loop here:



```
for (int i = 0; i < 10; i++)
{
    System.out.println(MediaFile.readString());
}
MediaFile.saveAndClose();
```

19. Experiment with the for loop so your code loops too many times. In other words, if you have eight songs defined in your data file, loop more than eight times. What is displayed after the algorithm runs out of text from the data file?

    null

Often, you know exactly how many times you want to loop, and the for loop is the best construct to use. Other times, you cannot predict the exact number of times you want to loop. As just demonstrated, you may not know how long your data file is, so you cannot predict the number of times to loop.

20. Use a different looping construct: **Learn to use the while loop**, and complete all of the activities.

    Done.

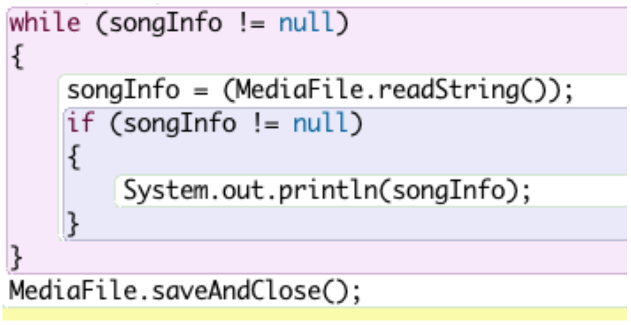
21. Convert your for loop to a while loop using the following condition:

```
        while (songInfo != null)
```

What is the **terminating condition** of the while loop, that is, the condition that ends the while loop? When does the terminating condition occur?

When the string is null.

Paste a snip of your while loop here:



```
while (songInfo != null)
{
    songInfo = (MediaFile.readString());
    if (songInfo != null)
    {
        System.out.println(songInfo);
    }
}
MediaFile.saveAndClose();
```

22. Now that you can easily read data from your data file, edit *mymedia.txt* directly and add some song titles and ratings using the existing syntax *song-title|rating*.

    Done.

23. Run LoopingMediaLib again to see the new data.

    Done.

24. The syntax of the data file is more computer friendly than it is user friendly. Use your knowledge of the string methods substring and indexOf to parse and reformat each songInfo string so that the output appears as show below.
    Title: *[song title]*
    Rating: *[song rating]*

    Done.

    Note: The bracket notation, such as *[song title]*, indicates the entire phrase including the square brackets should be replaced with data specific to your program, as follows:
    Title: The Twist
    Rating: 10

    Done.

Part IV: Other Data Representations

In Part IV you will change the way that data is stored in the media file so that instead of new lines in between each data entry, there will be instead a single character separating the items. You will then program a new algorithm to extract the data from the file since it is in a new format.

25. Small changes to a data algorithm can have major effects on data configuration. Find the line of code in the method writeString in MediaFile that calls the newline() method. Comment it out and write a pipe (|) to your file instead:

```
        // out.newLine();
         out.write("|");
```

I did something similar to this…

26. Run your original MediaLib program to regenerate the data file. Open the data file and describe how the data changed.

It's different, pipes and newlines are at different places.

27. To parse this new format, create a new FavoritesMediaLib class:

```
public class FavoritesMediaLib
{
    public static void main()
    {

    }
}
```

Skip 28 and 29… they seem pointless.

Part V: Convert Strings to Ints

You have parsed your data file and have retrieved data for all of your songs, including the rating. What if you want to find the highest rated song, or your least favorite song? You can convert from the string data type to an integer to compare ratings to one another. In this part of the activity, you will see how to loop over the songs to find the highest rating.

To convert from a string to an integer, you can use a new class, Integer. Similar to the Math.random() method you used in your SciFiName project, Java provides a static Integer.valueOf(String s) method that returns an integer value, passed in as the string parameter s. Note an important feature of this method: If String s cannot be parsed or interpreted as an integer value, your program will crash.

28. Write an algorithm that shows only your favorite songs:
    a. Choose the best looping construct.

       Done.

    b. Choose a rating value that indicates a favorite (e.g. 8 or higher), and show songs with only that rating or higher.

       Done.

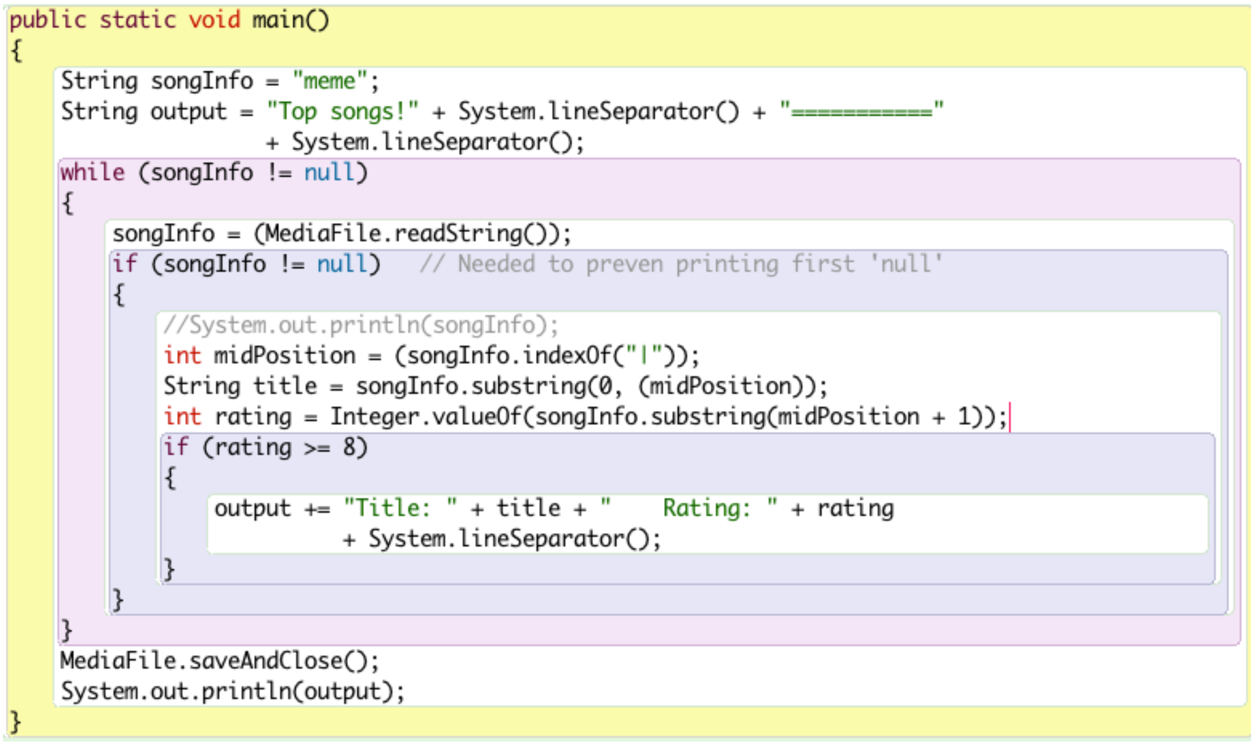
    c. Parse titles and ratings of songs separately; if you try to convert a title to an integer using something like Integer.valueOf(token), your program will crash.

       Done.

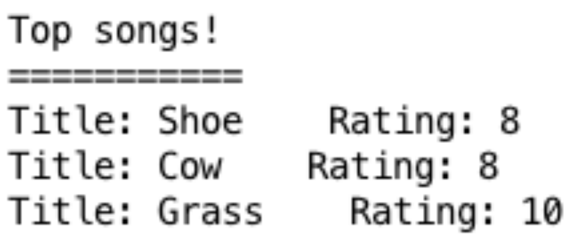
d. Show the output in a user-friendly format as shown with these Billboard hits : (Paste snips of the output and your method below)

My Favorite Songs

The Twist(10)
Smooth(9)
Mack the Knife(8)
Macarena(8)
Physical(9)
You Light Up My Life(10)
Hey Jude(9)



```
public static void main()
{
    String songInfo = "meme";
    String output = "Top songs!" + System.lineSeparator() + "==========="
                    + System.lineSeparator();
    while (songInfo != null)
    {
        songInfo = (MediaFile.readString());
        if (songInfo != null)   // Needed to preven printing first 'null'
        {
            //System.out.println(songInfo);
            int midPosition = (songInfo.indexOf("|"));
            String title = songInfo.substring(0, (midPosition));
            int rating = Integer.valueOf(songInfo.substring(midPosition + 1));
            if (rating >= 8)
            {
                output += "Title: " + title + "     Rating: " + rating
                          + System.lineSeparator();
            }
        }
    }
    MediaFile.saveAndClose();
    System.out.println(output);
}
```



```
Top songs!
===========
Title: Shoe    Rating: 8
Title: Cow    Rating: 8
Title: Grass    Rating: 10
```

Conclusion

1. Give two iteration examples, one where a for loop would be a good construct to use, and another where a while loop would be preferred. Explain why each iteration construct is best in each example.

```
while (String != null)
        {
                // Do something here
        }

for (int i =0; i < 10; i ++)
        {
                // Do something here
        }
```

For loops are more useful for when you know how many times you want to loop. While loops are better for when you do not know how many times you need to loop. Examples of while loops are: reading a file until a specific line, printing a string (letter by letter). For loop examples: reading fifty lines from a text file, printing numbers 1-50.

2. What major change in hardware prompted the development of Android OS 3.0, Honeycomb, and why did maintaining older versions like Gingerbread become extra cumbersome?

Tablets and more diversity in screen sizes.