Working with Classes

Movie Database Lab

# Main Objectives

* Level 1: Write a basic class with fields, constructor, getters, setters, and mutators.
* Level 2: Write a class which has a List as a field and requires manipulating a List and building an Array out of a subsection of the List.
* Level 3: Manipulate sections of Strings using String methods to automatically sub divide a String into specific lengths without breaking ‘proper’ words.

# Prior Knowledge

* Level 1: Familiarity with writing classes (fields, constructors, methods) and using Strings.
* Level 2: Familiarity with Lists and arrays.
* Level 3: Familiarity with String methods.

# The Lack of Greenfoot Background

Although this lab is written with Greenfoot to perform the visualizations, students do not need to use any of the Greenfoot API. All of the visualization of data on the screen has been pre-provided. Students will be writing classes with no Greenfoot requirements. This exhibits the qualities that occur in many industry practices, where one team works on the graphics and assumes the other team writes the classes correctly.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your Movie Database folder run the project file with the Greenfoot icon to begin.

# Demonstration

View a sample demonstration of what the Movie Database lab could look like here:

<https://www.greenfoot.org/scenarios/23658>

# Instructions

This lab is separated into three levels of difficulty. Each project independently has the same end results with differences only in how much a student must work on or how much prior knowledge the student needs to have. A student who finishes a higher-level part of this lab would not benefit from going to the lower-level parts.

* To make a new class in Greenfoot hit “Edit” then “New Class”

or type “Cntrl-n” with your keyboard.



## Movie Database Level 1

A Movie defines a class which stores basic information about a Movie including the title, the year of released, a description, and an average rating. The rating for a Movie may change over time as new ratings are added.

A Movie object can be constructed with five parameters: a String, an integer, a String, and two more integers. The first parameter specifies the title of the Movie. The second parameter specifies the year the Movie was released. The third parameter specifies a description of the Movie. The fourth parameter is the current number of times the Movie has been rated. The fifth parameter represents the current sum of all ratings.

A Movie object can also be constructed with three parameters: a String, an integer, a String, and two more integers. This second constructor should be the same as the first but default the current number of times the Movie has been rated to 0 and the current sum of all ratings to 0 as well.

A Movie has five methods:

addRating which takes in an integer and represents one new additional rating for the movie.

getRating which returns the current average of the movie’s rating as a double.

getTitle which returns the title of the movie as a String.

getYear which returns the year the movie was released as an integer.

getDescription which returns the description of the movie as a String.

The following example illustrates the behavior of Movie objects:

Movie m1 = new Movie( “AP Comp Sci A”, 2019,

“Teenagers have fun while taking an exam.”,

9, 90 );

System.out.println( m1.getYear() + “ “ + m1.getTitle() );

System.out.println( m1.getDescription() );

System.out.println( “Rated: “ + m1.getRating() );

m1.addRating( 7 );

System.out.println( “Adjusted rating: “ + m1.getRating() );

Movie m2 = new Movie( “Practice Exam”, 2015,

“Student brains stop working worldwide.” );

System.out.println( m1.getTitle() );

The output is shown below. Because m1 added a new rating of 7, the new average would be based off 10 votes with a new sum of 97, making an average of 9.7 for the movie.

2019 AP Comp Sci A

Teenagers have fun while taking an exam.

Rated: 10.0

Adjusted rating: 9.7

Practice Exam

Write the entire Movie class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be private. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the output shown).

If done correctly, the Greenfoot Movie Database Level 1 program should compile. You may need to open the MovieDatabase class and hit compile once for it to do so.

## Movie Database Level 2

* First complete all Movie Database Level 1 instructions above.

A MovieCollection defines a class which stores multiple Movies. The number of Movies may increase or decrease.

A MovieCollection object can be constructed with zero parameters which means it contains no initial Movies.

A MovieCollection has four methods:

* addMovie which takes in a Movie and adds the Movie to the collection.
* removeMovie which takes in a Movie and removes that Movie from the collection if the Movie is in the collection. If multiple Movie matches are found in the Collection, then any one of the Movies can be removed. If no matches are found then the MovieCollection should remain unchanged.
* getMovieSubset which takes in two integers. The first represents how many Movies to be returned and the second represents which index to start at. The first parameter can be any integer. The second parameter will always be greater than or equal to 0 and less than the number of Movies in the MovieCollection. The method should return a Movie array with the correct Movies inside it. If the first parameter is negative or zero no Movies should be in the returned Movie array. If the first parameter is greater than the number of Movies in the MovieCollection return only the number of Movies that are in the MovieCollection. See the examples below.
* size which returns how many Movies and in the MovieCollection.

The following example illustrates the behavior of MovieCollection objects:

Example 1:

MovieCollection c = new MovieCollection();

Movie a = new Movie( “A”, 1900, “Example 1”, 5, 20);

c.addMovie(a);

c.addMovie( new Movie( “B”, 1920, “Example 2”, 1, 5 ));

c.addMovie( new Movie( “C”, 1940, “Example 3”, 1, 1 ));

c.addMovie( new Movie( “D”, 1960, “Example 4”, 10, 90 ));

c.addMovie( new Movie( “E”, 1980, “Example 5”, 485, 2238 ));

c.addMovie( new Movie( “F”, 2000, “Example 6”, 2, 20 ));

System.out.println( c.size() );

c.removeMovie( a );

System.out.println( c.size() );

The output is shown below.

6

5

Example 2:

MovieCollection c2 = new MovieCollection();

C2.addMovie( new Movie( “A”, 1900, “Example 1”, 5, 20 ));

C2.addMovie( new Movie( “B”, 1920, “Example 2”, 1, 5 ));

C2.addMovie( new Movie( “C”, 1940, “Example 3”, 1, 1 ));

C2.addMovie( new Movie( “D”, 1960, “Example 4”, 10, 90 ));

C2.addMovie( new Movie( “E”, 1980, “Example 5”, 485, 2238 ));

C2.addMovie( new Movie( “F”, 2000, “Example 6”, 2, 20 ));

Returned Arrays

(Only showing titles, but actual

Array contents would be the full

Movies.)

Movie[] m1 = c2.getMovieSubset( 5, 0 ); {A, B, C, D, E}

Movie[] m2 = c2.getMovieSubset( 9, 2 ); {C, D, E, F, A, B}

Movie[] m3 = c2.getMovieSubset( 1, 3 ); {D}

Movie[] m4 = c2.getMovieSubset( 0, 2 ); {}

Movie[] m4 = c2.getMovieSubset( 3, 4 ); {E, F, A}

Movie[] m6 = c2.getMovieSubset( 4, 5 ); {F, A, B ,C}

Movie[] m7 = c2.getMovieSubset( -3, 0 ); {}

Write the entire MovieCollection class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be private. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the output shown).

If done correctly, the Greenfoot Movie Database Level 2 program should compile. You may need to open the MovieDatabase class and hit compile once for it to do so.

## Movie Database Level 3

* First complete all Movie Database Level 1 and 2 instructions above.

In this part of the lab you will be adding one method to the Movie class:

public String getFormattedDescription( int lineLength )

Write the Movie method getFormattedDescription which returns a formatted String with the Movie’s description broken into ‘proper’ subsets of length <= lineLength each separated by a new line character. Each subset of the returned String should use as close to lineLength number of characters as it can but should only split at spaces. If a particular substring of the description is longer than the lineLength, count it as a single subsection even if it is longer than lineLength.

Examples.

If the Movie description is:

“The 2016 APCSA exam question number 2 was Log Message.”

0123456789012345678901234567890123456789012345678901234567890123456

getFormattedDescription( 20 ) would return:

“The 2016 APCSA exam\nquestion number 2\nwas Log Message.”

getFormattedDescription( 10 ) would return:

“The 2016\nAPCSA exam\nquestion\nnumber 2\nwas Log\nMessage.”

getFormattedDescription( 5 ) would return:

“The\n2016\nAPCSA\nexam\nquestion\nnumber\n2 was\nLog\nMessage.”

If done correctly, the Greenfoot Movie Database Level 3 program should compile. You may need to open the MovieDatabase class and hit compile once for it to do so.

# Extensions and Modifications

* The given Greenfoot code in the World and Actors is not overly complicated and could have some or all of its sections erased and be finished by students instead.
* It is possible for students to self-learn how to pull any image off of an online website to allow any movie to be added to their program.
* This project currently runs off of Greenfoot, but it could be used off of standard input/out or a different image program. Students could be tasked with writing the input/output using System.in and Scanner to test the classes themselves, while still being able to use the exact same code in the given Greenfoot project. Students could also be tasked with using JavaFX or Swing to create a GUI that can be run outside of Greenfoot’s IDE.
* The current placement of objects uses quite a few “magic numbers”. The screen is thus not resizable well. Students could be tasked with adjusting the numbers to be based off of math calculations of the world’s width and height, the sizes of the MovieIcons, etcetera. Students would then need to demonstrated changing the size of things and still having all visual items on the screen displaying in correct locations.
* The current instructions for part 3 have students ripping the String into subsections and placing new line characters between each new line. This could be modified, and is arguably better to be done so, in such a way that the String is broken down into subsections and returned as a List of Strings instead of being returned as a single String. Any objects uses this method would now be able to decide how it wants to use the subsections, would they like to use new lines between or would they like some other option.