



## Project 02: Mooglee

A simple movie search system

Due: Sunday April 1, 2018 11:55PM

### I. Learning Objectives:

Most people use search engines, like Google, almost all the time. In this project, you will learn how to implement a “simple” search system to search for movies. Upon finishing the project, we expect you to be able to

1. Get a glimpse of how OOP can be applied to real-world problems
2. Use Java Collection including List, Set, and Map collection type
3. Access the data in the file and parse them to create objects
4. Learn how to use regular expressions and simple string manipulation techniques.
5. Develop searching and sorting algorithms
6. Be inspired and enjoy coding with Java =D

Please note that this project is a building block for your next project, which you will build a recommendation system for movies in Mooglee.

### II. Introduction:

A search engine application is a tool that helps users to find a specific content that matches with users’ information needs. In this project, your task is to implement a search engine application for movies. Some part of the system is already provided for you in the project package.

## Technical Definitions:

*Movie*: A movie is a tuple of (`mid`, `title`, `year`, `tags`, `ratings`). `mid` is the ID of the movie, always a positive integer. `title` is the String title of the movie. `year` is a positive integer representing the year in which this movie was released. `tags` is a set of String relevant tags that describe this movie. Finally, `ratings` is the set of ratings that this movie has been rated by users. Ratings are stored in a `Map<uid, Rating>` data structure for fast lookup. Noted that `uid` is the ID of the user.

*Rating*: A rating is a tuple of (`uid`, `mid`, `score`, `timestamp`). `uid` and `mid` are the IDs of the rating user and the rated movie, respectively. `score` is a double numeric value. If given, a rating score can range from [0.5,5] inclusive. A rating score of 0 or negative values implies that no rating information of the user *uid* and the movie *mid* is available (the user may never have rated this movie or the rating is made unavailable). `timestamp` is a long value indicating the time at which this rating was given. What is a timestamp?<sup>1</sup>

*User*: A user is a tuple of (`uid`). `uid` is the ID of the user, always a positive integer.

## Movie file format:

A movie file (e.g. `movies.csv`) stores the meta-information about all the movies. Except for the first line which is the table header, each line in the movie file has one of the following formats:

`<mid>,<title> (<year>),<tag_1>|<tag_2>|<tag_3>|...|<tag_n>`  
*//title does not contain ‘ ’*

`<mid>,"<title> (<year>)",<tag_1>|<tag_2>|<tag_3>|...|<tag_n>`  
*//title contains ‘ ’*

For example,

```
movieId,title,genres
22,Copycat (1995),Crime|Drama|Horror|Mystery|Thriller
23,Assassins (1995),Action|Crime|Thriller
24,Powder (1995),Drama|Sci-Fi
54,"Big Green, The (1995)",Children|Comedy
25,Leaving Las Vegas (1995),Drama|Romance
```

Your system needs to parse this file to load the movies into the memory. The movies are stored in a `Map` structure where they can be looked up using their `mid`'s.

---

<sup>1</sup> See: <https://en.wikipedia.org/wiki/Timestamp>

Hint: You may find regex and String.split() useful to parse this file.

### **Rating file format:**

A rating file (e.g., ratings.csv) lists individual ratings. Rating generated by the users of the same movie can be grouped together, and stored in a Map structure of the same Movie object. Except for the first line which contains the table headers, each line in the user file has the following format:

<uid>,<mid>,<rating>,<timestamp>

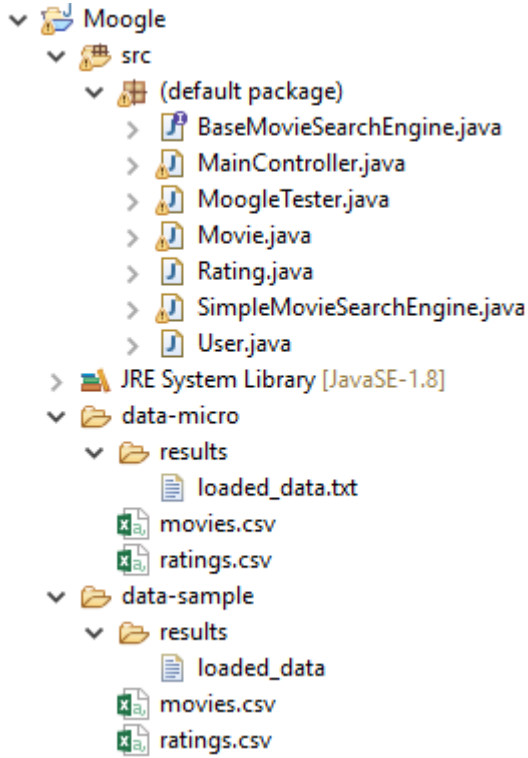
For example,

```
userId,movieId,rating,timestamp
6,94864,4.5,1348693600
6,78039,3.0,1348693590
6,76251,3.0,1348693953
```

Hint: You may find the method String.split() useful when parsing a rating file.

### **III. Instructions**

In the project source code package, **package-moogle**, you will find the following files:

	<p>The <b>src</b> folder contains necessary source code files:</p> <p>Movie.java Rating.java User.java BaseMovieSearchEngine.java MoogleTester.java MainController.java</p> <p>You must put the above files in the <b>default package</b>!</p> <p>There are two sets of data folders: <b>data-micro</b> and <b>data-sample</b> folder. Each folder has movies.csv and ratings.csv file. Data in data-micro folder is a subset of data in data-sample folder.</p>
---	--

You must **NOT** modify BaseMovieSearchEngine.java, MooglerTester.java and the contents in data folders. You must complete the code in Movie.java, Rating.java, and User.java. In addition, you have to implement a new class called SimpleMovieSearchEngine in SimpleMovieSearchEngine.java that **implements** the interface BaseMovieSearchEngine. You will find the description of each method in BaseMovieSearchEngine.java.

Note that your methods should appropriately handle corner cases such as null references, out-of-range values, NaN/Inf values, division by 0, etc.

### **Test cases:**

Two ways to debug your code is provided.

First, you can use MooglerTester class to test your program. This program only uses files in data-micro folder. The expected output is shown below

```
[mid: 32:Twelve Monkeys (a.k.a. 12 Monkeys) (1995) [Sci-Fi, Thriller, Mystery]] ->
avg rating: 4.0
  [uid: 1 mid: 32 rating: 4.0/5 timestamp: 1217896246]
[mid: 111360:Lucy (2014) [Action, Sci-Fi]] -> avg rating: 2.5
  [uid: 21 mid: 111360 rating: 2.5/5 timestamp: 1428011135]
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
  [uid: 128 mid: 100326 rating: 4.0/5 timestamp: 1420011101]
  [uid: 475 mid: 100326 rating: 3.5/5 timestamp: 1388005962]
[mid: 101864:Oblivion (2013) [Action, Sci-Fi, Adventure, IMAX]] -> avg rating: 3.5
  [uid: 128 mid: 101864 rating: 3.5/5 timestamp: 1420251474]
[mid: 3018:Re-Animator (1985) [Sci-Fi, Horror, Comedy]] -> avg rating: 2.5
  [uid: 475 mid: 3018 rating: 2.5/5 timestamp: 1379678927]
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5
  [uid: 128 mid: 100108 rating: 2.5/5 timestamp: 1445957961]
[mid: 109487:Interstellar (2014) [Sci-Fi, IMAX]] -> avg rating: 4.0
  [uid: 21 mid: 109487 rating: 4.0/5 timestamp: 1428010919]
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
  [uid: 128 mid: 107406 rating: 1.5/5 timestamp: 1419657454]
[mid: 16:Casino (1995) [Drama, Crime]] -> avg rating: 3.0
  [uid: 1 mid: 16 rating: 3.0/5 timestamp: 1217898888]
[mid: 19:Ace Ventura: When Nature Calls (1995) [Comedy]] -> avg rating: 3.5
  [uid: 21 mid: 19 rating: 3.5/5 timestamp: 1428095348]
[mid: 3030:Yojimbo (1961) [Action, Adventure]] -> avg rating: 4.0
  [uid: 475 mid: 3030 rating: 4.0/5 timestamp: 1305325978]
[mid: 24:Powder (1995) [Sci-Fi, Drama]] -> avg rating: 1.5
  [uid: 1 mid: 24 rating: 1.5/5 timestamp: 1217895807]
[mid: 106489:Hobbit: The Desolation of Smaug, The (2013) [Adventure, Fantasy, IMAX]]
-> avg rating: 4.0
  [uid: 21 mid: 106489 rating: 4.0/5 timestamp: 1428011496]
[mid: 108729:Enemy (2013) [Thriller, Mystery]] -> avg rating: 3.5
  [uid: 128 mid: 108729 rating: 3.5/5 timestamp: 1419294410]
[mid: 99992:Shadow Dancer (2012) [Drama, Thriller, Crime]] -> avg rating: 3.0
  [uid: 475 mid: 99992 rating: 3.0/5 timestamp: 1385062503]
[mid: 3033:Spaceballs (1987) [Sci-Fi, Comedy]] -> avg rating: 3.75
  [uid: 21 mid: 3033 rating: 4.0/5 timestamp: 1428014072]
  [uid: 475 mid: 3033 rating: 3.5/5 timestamp: 1305331090]
[mid: 108188:Jack Ryan: Shadow Recruit (2014) [Action, Drama, Thriller, IMAX]] -> avg
rating: 3.5
  [uid: 128 mid: 108188 rating: 3.5/5 timestamp: 1419746244]
```

```

[mid: 99996:It's a Disaster (2012) [Drama, Comedy]] -> avg rating: 3.5
  [uid: 475 mid: 99996 rating: 3.5/5 timestamp: 1378753434]
[mid: 108190:Divergent (2014) [Sci-Fi, Adventure, Romance, IMAX]] -> avg rating: 3.25
  [uid: 128 mid: 108190 rating: 2.5/5 timestamp: 1419707507]
  [uid: 21 mid: 108190 rating: 4.0/5 timestamp: 1428011523]
*****
Total number of movies: 19
Total number of ratings: 22
***** RESULTS *****
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
Search by exact title = stand up guys
1 movies found!
***** RESULTS *****
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
Search by approximate title = and
1 movies found!
***** RESULTS *****
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
[mid: 3018:Re-Animator (1985) [Sci-Fi, Horror, Comedy]] -> avg rating: 2.5
[mid: 19:Ace Ventura: When Nature Calls (1995) [Comedy]] -> avg rating: 3.5
[mid: 3033:Spaceballs (1987) [Sci-Fi, Comedy]] -> avg rating: 3.75
[mid: 99996:It's a Disaster (2012) [Drama, Comedy]] -> avg rating: 3.5
Search by tag = Comedy
5 movies found!
***** RESULTS *****
[mid: 101864:Oblivion (2013) [Action, Sci-Fi, Adventure, IMAX]] -> avg rating: 3.5
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
[mid: 106489:Hobbit: The Desolation of Smaug, The (2013) [Adventure, Fantasy, IMAX]]
-> avg rating: 4.0
[mid: 108729:Enemy (2013) [Thriller, Mystery]] -> avg rating: 3.5
Search by year = 2013
5 movies found!
***** RESULTS *****
[mid: 108729:Enemy (2013) [Thriller, Mystery]] -> avg rating: 3.5
[mid: 106489:Hobbit: The Desolation of Smaug, The (2013) [Adventure, Fantasy, IMAX]]
-> avg rating: 4.0
[mid: 101864:Oblivion (2013) [Action, Sci-Fi, Adventure, IMAX]] -> avg rating: 3.5
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
Sorted by title in ascending order
5 movies found!
***** RESULTS *****
[mid: 106489:Hobbit: The Desolation of Smaug, The (2013) [Adventure, Fantasy, IMAX]]
-> avg rating: 4.0
[mid: 101864:Oblivion (2013) [Action, Sci-Fi, Adventure, IMAX]] -> avg rating: 3.5
[mid: 108729:Enemy (2013) [Thriller, Mystery]] -> avg rating: 3.5
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
Sorted by rating in descending order
5 movies found!
***** RESULTS *****
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
[mid: 99996:It's a Disaster (2012) [Drama, Comedy]] -> avg rating: 3.5
Advance search by approximate tag = Comedy and year = 2012
2 movies found!
***** RESULTS *****
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75
Advance search by approximate title = and and tag = Comedy
1 movies found!
***** RESULTS *****
Advance search by approximate title = and, tag = Comedy, and year = 2016
0 movies found!

```

Second, to test your code, produce a runnable jar file MainController.jar from MainController class. Open a Command Prompt (Windows), Terminal (Linux/Mac OS), and "cd" to the directory where MainController.jar and test cases are located. You can test your program in different functions by providing different input parameter. For example,

1. Loading movies and ratings: load <movie\_filename> <rating\_filename>  
Ex. java -jar MainController.jar load movies.csv ratings.csv

```
*****  
Total number of movies: 19  
Total number of ratings: 22
```

2. Searching movies by title (exact matching): search <movie\_filename>  
<rating\_filename> -e <title>  
Ex. java -jar MainController.jar search movies.csv ratings.csv -e "stand up guys"

```
***** RESULTS *****  
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75  
Keyword -> title (exact) = stand up guys  
1 movie(s) found
```

3. Searching movies by title (approximate matching): search <movie\_filename>  
<rating\_filename> -a <title>  
Ex. java -jar MainController.jar search movies.csv ratings.csv -a "stand"

```
***** RESULTS *****  
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75  
Keyword -> title (approximate) = stand  
1 movie(s) found
```

4. Searching movies by tag: search <movie\_filename> <rating\_filename> -t <tag>  
Ex. java -jar MainController.jar search movies.csv ratings.csv -t Comedy

```
***** RESULTS *****  
[mid: 100326:Stand Up Guys (2012) [Crime, Comedy]] -> avg rating: 3.75  
[mid: 3018:Re-Animator (1985) [Sci-Fi, Horror, Comedy]] -> avg rating: 2.5  
[mid: 19:Ace Ventura: When Nature Calls (1995) [Comedy]] -> avg rating: 3.5  
[mid: 3033:Spaceballs (1987) [Sci-Fi, Comedy]] -> avg rating: 3.75  
[mid: 99996:It's a Disaster (2012) [Drama, Comedy]] -> avg rating: 3.5  
Keyword -> tag = Comedy  
5 movie(s) found
```

5. Searching movies by year: search <movie\_filename> <rating\_filename> -y <year>  
Ex. java -jar MainController.jar search movies.csv ratings.csv -y 2013

```
***** RESULTS *****  
[mid: 101864:Oblivion (2013) [Action, Sci-Fi, Adventure, IMAX]] -> avg rating: 3.5  
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5  
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5  
[mid: 106489:Hobbit: The Desolation of Smaug, The (2013) [Adventure, Fantasy, IMAX]]  
-> avg rating: 4.0  
[mid: 108729:Enemy (2013) [Thriller, Mystery]] -> avg rating: 3.5  
Keyword -> year = 2013  
5 movie(s) found
```

6. Advance search movie: search+ <movie\_filename> <rating\_filename> [-a]  
[<title>] [-t] [<tag>] [-y] [<year>]

Ex. java -jar MainController.jar search+ movies.csv ratings.csv -a "all" -t Sci-Fi

```
***** RESULTS *****
[mid: 3033:Spaceballs (1987) [Sci-Fi, Comedy]] -> avg rating: 3.75
Keywords -> ,title = all ,tag = Sci-Fi
1 movie(s) found
```

Ex. java -jar MainController.jar search+ movies.csv ratings.csv -a "er" -t Drama -y 2013

```
***** RESULTS *****
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
Keywords -> ,title = er ,tag = Drama ,year = 2013
1 movie(s) found
```

Ex. java -jar MainController.jar search+ movies.csv ratings.csv -a "er" -y 2013

```
***** RESULTS *****
[mid: 100108:Parker (2013) [Thriller, Crime]] -> avg rating: 2.5
[mid: 107406:Snowpiercer (2013) [Action, Sci-Fi, Drama]] -> avg rating: 1.5
Keywords -> ,title = er ,year = 2013
2 movie(s) found
```

After the results are displayed, the system will ask you to choose the next step. You can choose to sort the result only after you already did searching.

```
What do you want to do next?
type 't' to sort the result by title in ascending order,
type 'T' to sort the result by title in descending order,
type 'r' to sort the result by movie's average rating in ascending order,
type 'R' to sort the result by movie's average rating in descending order,
type 's' follow by the search criteria (-e/-a/-t/-y) and keyword to search movies
again with normal search criteria,
type 'S' follow by the search criteria and list of keywords to search movies again
with advanced search criteria
type 'q' to quite the program.
```

#### IV. Coding Style Guideline:


It is important that you follow this coding style guideline strictly, to help us with the grading and for your own benefit when working in group in future courses. Failing to follow these guidelines may result in a deduction of your project scores.

- Write your name, student ID, and section on top of every code file that you submit.

- Comment your code, especially where you believe other people will have trouble understanding, such as purposes of each variable, loop, and chunk of code. If you implement a new method, make sure to put an overview comment at the beginning of each method that explains 1) Objective, 2) Inputs (if any), and 3) Output (if any).
- Do not put your code in new packages. Put everything in the default package.

## VI. Grading:

Grade for this project will be given in percentage basis (max 100 points, without bonus). Here's the tentative breakdown of the criteria:

If you submit 	5 points
Follow the coding style	5 points
Load all the data correctly	20 points
Outputs – get correct search result	30 points
Outputs – able to sort the result	20 points
Other specific method functionalities	20 points
Total	100 points

**Optional:** Bonus for creativity 10 points (max)

You are encouraged to implement your own cool feature (~be creative~) such as creating a GUI/Web interface for your search engine using JAVA, creating a new function to store new movies or new reviews by a user, developing a new sorting algorithm to sort by relevance, implementing a model to find the similar movie, and etc. To qualify for these bonus points, you also have to submit a report describing your new feature.

## VI. Submission

It is important that you follow the submission instructions to enable your code and output files to be graded by the auto grader. Failing to do so may result in a deduction and/or fault evaluation of your project.



- Zip all of your source code files in a single file namely “**P2\_studentID.zip**” (without “ ”). E.g. P2\_6088999.zip.
- To claim your bonus points for your extra feature, you **MUST** submit a short report in PDF (less than five pages) describing your new feature. If you do not submit the report, your new feature will not be graded.
- Upload your zip file in the My Courses system under **Project02 Submission** by the deadline.
- **RECHECK YOUR SUBMISSION.** Always re-check what you just submitted is actually what you wanted to submit. Any submission received after the deadline will be considered late submission and will receive the same penalty as one—no excuses.

Late submission will suffer a penalty of 20% deduction of the actual scores for each late day.

You can keep resubmitting your solutions, but the only latest version will be graded. *Even one second late is considered late.*

## VII. Suggestions

1. Though Mooglee is a fairly simple movie search system, the small details can overwhelm you during the course of implementation. Our suggestion is to incrementally implement and test one method at a time, to see if it behaves as expected. Sometimes, you may need to write your own test cases to test specific functionalities, which is much encouraged. It is discouraged to implement everything and test it all at once. Our suggestion is to implement/test the methods in the following order:

```
- public Map<Integer,Movie> loadMovies(String movieFilename);
- public void loadRating(String ratingFilename);
- public void loadData(String movieFilename, String ratingFilename);
- public Map<Integer, Movie> getAllMovies();
- public List<Movie> searchByTitle(String title, boolean exactMatch);
- public List<Movie> searchByTag(String tag);
- public List<Movie> searchByYear(int year);
- public List<Movie> searchMovies(String title, String tag, int year);
- public List<Movie> sortByTitle(List<Movie> unsortedMovies, boolean asc);
- public List<Movie> sortByRating(List<Movie> unsortedMovies, boolean asc);
```

2. Map data structure<sup>2</sup> (e.g. HashMap) allows fast lookup on your data. For example, getting Movie objects using `mid`'s as keys takes only  $O(1)$  runtime complexity.
3. Use Set data structure<sup>3</sup> (e.g. HashSet) whenever your data can be stored using a set representation.
4. Even though this project is much easier than Project 1, you have only a few weeks to finish it. Therefore, you should start early so you have enough time to cope with unforeseen situations.
5. Use Java JDK 1.8. Some of the functionalities may not be available in the older versions.

### **Bug Report and Specs Clarification:**

Though not likely, it is possible that our solutions may contain bugs. A bug in this context is not an insect, but error in the solution code that we implemented to generate the test cases.

Hence, if you believe that your implementation is correct, yet yields different results, please contact us immediately so proper actions can be taken. Likewise, if the project specs contain instructions that are ambiguous, please notify us immediately.

### **Need help with the project?**

If you have questions about the project, please first post them on the forum on My Courses, so that other students with similar questions can benefit from the discussions. The tutors can also answer some of the questions and help to debug trivial errors. If you still have questions or concerns, please make an appointment with one of the instructors. We do not debug your code via email. If you need help with debugging your code, please come see us. Consulting ideas among friends is encouraged; however, the code must be written by yourself without looking at others' code. (See next section.)

---

<sup>2</sup> <https://docs.oracle.com/javase/7/docs/api/java/util/Map.html>

<sup>3</sup> <https://docs.oracle.com/javase/7/docs/api/java/util/Set.html>

## Academic Integrity

Don't get bored about these warnings yet. But please, please do your own work. Your survival in the subsequent courses heavily depends on the programming skills that you harvest in this course. Though students are allowed and encouraged to discuss ideas with others, the actual solutions must be written by themselves without ***being dictated or looking at others' code***. Collaboration in writing solutions is not allowed, as it would be unfair to other students. It is better to submit a broken program that is a result of your own effort than taking somebody else's work for your own credit! Students who know how to obtain the solutions are encouraged to help others by guiding them and teaching them the core material needed to complete the project, rather than giving away the solutions. *\*\*You can't keep helping your friends forever, so you would do them a favor by allowing them to be better problem solvers and life-long learners. \*\** If you get caught cheating, serious actions will be taken!