Movie Manager
Proposal

Sanjana Cheerla
CSC316: Data Structures & Algorithms
scheerl@ncsu.edu

North Carolina State University
Department of Computer Science

September 17th, 2020

System Test Plan

For my Test Cases I will be using the following two files:

1. movieRecord.txt

```
ID,Title,Year,Runtime (Minutes),Genre
MAAS1,Guardians of the Galaxy,2014,121,Action,Adventure,Sci-Fi
MADS103,The Martian,2015,144,Adventure,Drama,Sci-Fi
MAFF344,Pete's Dragon,2016,102,Adventure,Family,Fantasy
MAAF6,The Great Wall,2016,103,Action,Adventure,Fantasy
MBDH12,Hidden Figures,2016,127,Biography,Drama,History
MAAF51,Star Wars: Episode VII - The Force
Awakens,2015,136,Action,Adventure,Fantasy
MCDM7,La La Land,2016,128,Comedy,Drama,Music
MAAS681,The Hunger Games: Mockingjay - Part
1,2014,123,Action,Adventure,Sci-Fi
MADF121,Miss Peregrine's Home for Peculiar

Children,2016,127,Adventure,Drama,Family
```

2. watchRecord.txt

```
Movie ID,Watch Date,Minutes Watched
MAAS1,09/10/2019,24
MAFF344,05/01/2019,11
MADS103,07/05/2019,120
MAFF344,03/04/2020,2
MBDH12,09/01/2020,125
MAAF6,08/12/2020,54
MAFF344,02/05/2020,45
MAAF51,05/04/2020,130
MAFF344,02/04/2020,102
```

Test ID	Description	Expected Results	Actual Results
testInvalidMovieRecordFile (unexpected Input)	Preconditions: - PackFlixUI has loaded successfully - The file x.txt does not exist Steps: 1. Click “Load movie record” Browse to select x.txt file 2. Click submit	The software does not crash and instead reprompts the user to specify a new file in a popup window.	matches expected results
testLoadValidFiles (equivalence classes)	Preconditions: - PackFlixUI has loaded successfully - The file movieRecord.txt does exist	The Movie Record file loads and displays the following 9 movies - `Guardians of the Galaxy,2014` - `The Martian,2015`	matches expected results

	- The file watchRecord.txt does exist Steps: 1. Click “Load movie record” 2. Browse to select movieRecord.txt file 3. Click submit 4. See expected results 5. Click “Load watch record” 6. Browse to select watchRecord.txt file 7. Click submit	- Pete's Dragon,2016, - The Great Wall,2016 - Hidden Figures,2016 - Star Wars: Episode VII - The Force Awakens,2015 - La La Land,2016 - The Hunger Games: Mockingjay - Part 1,2014 - Miss Peregrine's Home for Peculiar Children,2016 The Watch Record file loads and displays the following 9 records: - MAAS1,09/10/2019,24 - MAFF344,05/01/2019,11 - MADS103,07/05/2019,120 - MAFF344,03/04/2020,2 - MBDH12,09/01/2020,125 - MAAF6,08/12/2020,54 - MAFF344,02/05/2020,45 - MAAF51,05/04/2020,130 - MAFF344,02/04/2020,102	
testBoundaryFreqStreamedMovies	Preconditions: - PackFlixUI has loaded successfully	The Report is generated with the following information:	matches expected results

(boundary values)	- The files movieRecord.txt and watchRecord.txt load properly according to the test testLoadValidFiles Steps: 1. Click view a report of the most frequently watched movies 2. Enter 1 in the box to how many movies you wish to see in the report 3. Click submit	`The 1 most frequently watched movies [ Pete's Dragon (2016) ]`	
testInvalidFreqStreamedMovies (unexpected input)	Preconditions: - PackFlixUI has loaded successfully - The files movieRecord.txt and watchRecord.txt load properly according to the test testLoadValidFiles Steps: 1. Click view a report of the most frequently watched movies 2. Enter 0 in the box to how many movies you wish to see in the report 3. Click submit	The software does not crash and tells the user to `"Please enter a number > 0."`	matches expected results
testInvalidPercentUnfinishedMovies (unexpected Input)	Preconditions: - PackFlixUI has loaded successfully - The files movieRecord.txt and watchRecord.txt load properly according to the test testLoadValidFiles Steps: 1. Click view a report the movies watched for a specific percentage 2. Enter -100 3. Click submit	The software does not crash and tells the user to `"Please enter a percentage completion between 1 and 100."`	matches expected results

Algorithm Design

```
Algorithm buildMovieMap(L)
Input L, a list of movie records
Output M, a map of movies structured as MovieID -> Movie record
//create an empty map for movies
movies <- empty Map
//go through movie records and map movies with their record
for i <- 0 to L.size - 1 do
      //put the movies into the map
      movies.put(L.get(i).id(), L.get(i).record())
//return map
return movies
```

```
Algorithm loadWatchDates(M, W)
Input M, a map of ID -> Movie record
    W, a list of Watch records
Output a Map of movie watch dates structured as MovieID -> List of Watch
records for the movie
//create an empty map for movies
movies <- empty Map
//loop through Map
for i <- 0 to M.size - 1 do
      //initialize empty list for watch records
      l <- empty List
      //loop through watch records adding matching records with movies to
      //the list
      for j <- 0 to W.size - 1 do
            //check to see if the movie has the same id as the watch record
            if(M.get(i).id() = W.get(j).id()) then
                  //add watch record to movie watch record list
                  l.addLast(W.get(j).record())
      //put the list of movies into the respective area into the map
      movies.put(M.get(i).id), l)
//return map
return movies
```

```
Algorithm loadWatchDuration(M, W, t)
Input M, a map of ID -> Movie record
    W, a list of Watch records
    t, a threshold percentage
Output a Map of movies with watch durations less than the threshold, t,
structured as MovieID -> watch percentage
//create an empty map for movies
movies <- empty Map
//loop through Map
for i <- 0 to M.size - 1 do
      //initialize time
      time <- -1
      //loop through the watch records for the movie and find the max time
      //watched
      for j <- 0 to W.size - 1 do
            //check to see if the movie has the same id as the watch record
            //time and if the current time is less than the time on record
            if (t < W.get(j).time() AND M.get(i).id() = W.get(j).id()) then
                  //set the max time between watch record and time to time
                  time = max(time, watchRecord.time())
      //if the movie is watched
      if(time > -1) then
            //then put the movie into the map
            movies.put(M.get(i).id), time)
//return map
return movies
```

Data Structures

For this project I will be using the following abstract data type:

- **List**
  I will use a list as my abstract data type because I can easily put all of the necessary objects such as Movie and put the objects into a list.
- **Map**
  I will use map to store the movie ID as keys and values as mentioned in the algorithms above

I will be using the following data structures:

- **HashMap**
  I will be using a Map data structure to store movies with movie records, movies with movie records, and movies with watch percentages. A map is appropriate for this type of data because there will not be multiple types of movies and each movie will have a unique id. The put and get operations in a map is also O(1) runtime.

- **Array-Based List**
  I want the array to dynamically resize based on the number of items in the list. A list is easily accessible with **get()** methods, **addLast(),** and **size()** methods which are O(1) run time. The array-based list will be used when information is read from the files, and when a movie is mapped with a list of the watch records for that movie (implemented in **Algorithm loadWatchDates(M, W)**).

I will be using the following sort algorithm:

- **Mergesort**
  I want to use merge sort because the average run time for this algorithm is O(N log N), and the worst-case performance is also O(N log N). This is helpful because the data needs to be organized in an ascending fashion by alphabetical order for when the user wants to view a report of the movies that fall below a specified percent completed. Also, the data needs to be sorted in descending order by date when the user wants a report of what dates a specific movie was streamed. I will sort the data overall according to the Movie id in an ascending manner.

Algorithm Analysis

```
Algorithm buildMovieMap(L)
Input L, a list of movie records
Output M, a map of movies structured as MovieID -> Movie record
1  movies <- empty Map
2  for i <- 0 to L.size - 1 do
3     movies.put(L.get(i).id(), L.get(i).record)
4  return movies
```

The for loop(line 2) has a run time of O(n) because it is iterating through the list of movie records. The put method for the map in line 3 has a run time of O(1), which is constant. So the overall average run time for the algorithm `buildMovieMap(L)` is O(n).

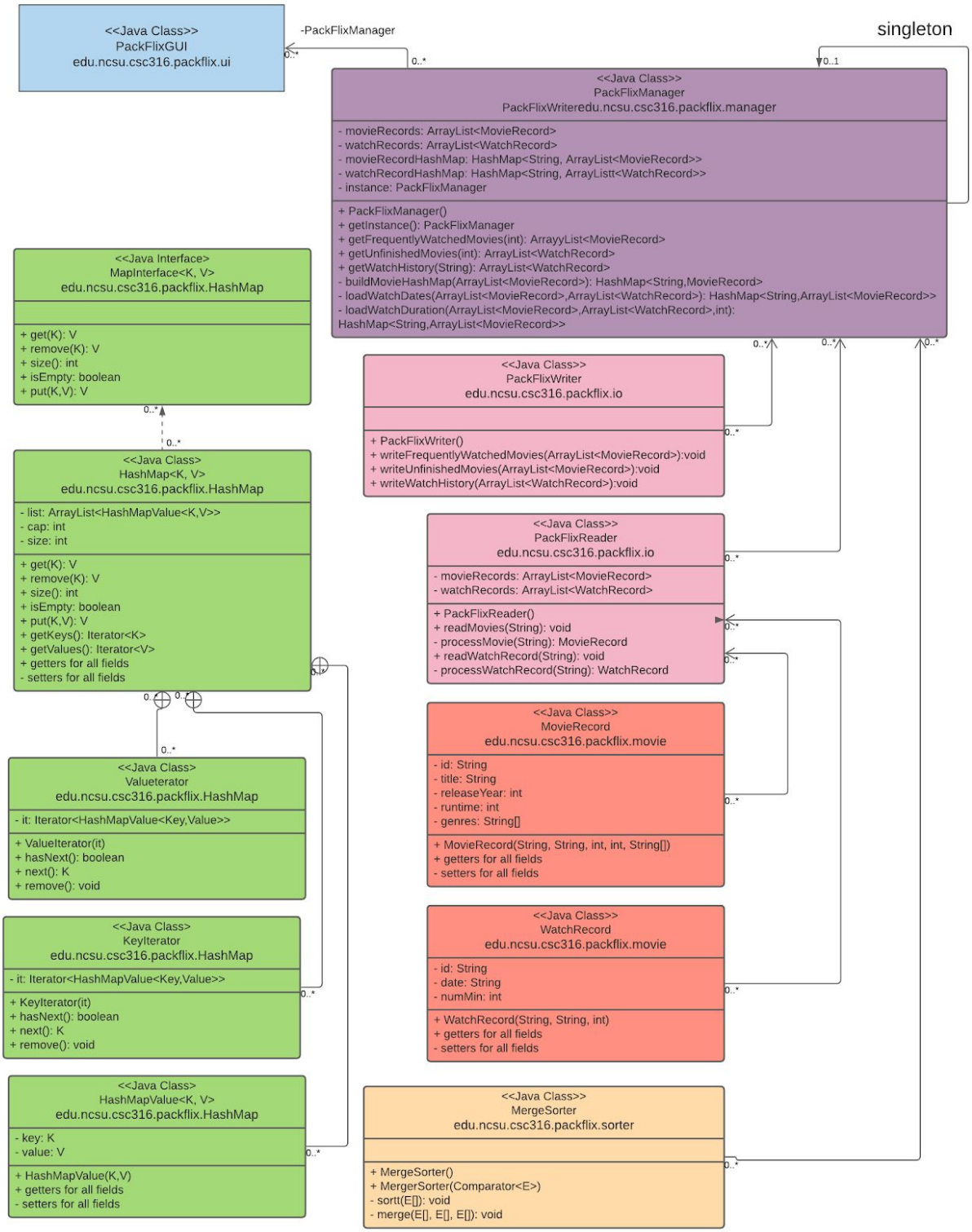
```
Algorithm loadWatchDates(M, W)
Input M, a map of ID -> Movie record
    W, a list of Watch records
Output a Map of movie watch dates structured as MovieID -> List of Watch
records for the movie
1  movies <- empty Map
2  for i <- 0 to M.size - 1 do
3     l <- empty List
4     for j <- 0 to W.size - 1 do
5           if(M.get(i).id() = W.get(j).id()) then
6                 l.addLast(W.get(j).record())
7     movies.put(M.get(i).id(), l)
8  return movies
```

The for loop(line 2) has a run time of O(n) because it is iterating through the map of movie records. The second for loop(line 4) also has a run time of O(n) because it is iterating through a list of watch records. The `addLast(element)` method in line 6 has a run time of O(1), which is constant. So the overall average run time for the algorithm `loadWatchDates(M, W)` is O(n^2).

```
Algorithm loadWatchDuration(M, W, t)
Input M, a map of ID -> Movie record
    W, a list of Watch records
    t, a threshold percentage
Output a Map of movies with watch durations less than the threshold, t,
structured as MovieID -> watch percentage
1  movies <- empty Map
2  for i <- 0 to M.size - 1 do
3     time <- -1
4     for j <- 0 to W.size - 1 do
5         if (t < W.get(j).time() AND M.get(i).id() = W.get(j).id()) then
6                 time = max(duration, watchRecord.time())
7     if(time > -1) then
8           movies.put(M.get(i).id(), time)
9  return movies
```

The for loop(line 2) has a run time of O(n) because it is iterating through the map of movie records. The second for loop(line 4) also has a run time of O(n) because it is iterating through a list of watch records. The `max(a, b)` method in line 6 has a run time of O(1), which is constant. The `put(k,v)` method in line 8 has a run time of O(1), which is constant. So the overall average run time for the algorithm `loadWatchDuration(M, W, t)` is O(n^2).

Software Design



According to my software design shown above, I am making use of the singleton design pattern and the MVC design pattern. I use the singleton pattern because there should only be one instance of PackFlixManager so that the data does not get corrupted or duplicated.

I used the MVC design pattern to split up the PackFlix project into three parts which makes my design have high cohesion as well as a clear organization for programming and testing. In my design, the model classes are object classes like MovieRecord, WatchRecord, etc. The view is the GUI class, and the controller is the Manager class which controls the PackFlix Instances using the singleton design pattern mentioned above/