Title: Investigating Gender Imbalance in Hollywood Movies Using Various Feminist Tests
Team Members: Tori Stiegman, Veronica Lin, Crystal Luo
Special Thanks: Prof. Smaranda for helping us! And thanks to ourselves for starting this project early and being really productive and collaborative as a team!

Introduction

In this project, we examined the gender imbalances in Hollywood movies. We set out to examine how "feminist" a set of Hollywood movies were based on the representation of females in the movies' casts, crew members, the way movies portray women, and most importantly, the inclusion of colored females. The way we assessed each criteria was based on what current women in the film industry believed to be the "next" Bechdel Test. After looking carefully at all these tests, **our group's definition of a "feminist" movie is one movie that passes all the following tests: the Peirce Test, Feldman Score Test, Rees Davies Test, and Waithe Test.** Below are the criteria for each test we chose:

Peirce Test -- a movie passes if…

- There's a female character who is a protagonist or antagonist with her own story
- The female lead has dimension and exists authentically with needs and desires that she pursues through dramatic action
- And the audience can empathize with or understand the female lead's desires and actions

The Feldman Score -- a movie passes with a score of five or higher...

- 2 points for a female writer or director
- 1 point for a female composer or director of photography
- 1 point for three female producers or three female department heads
- 1 point for a crew that's 50 percent women
- 2 points if there's a female protagonist who determines story outcomes
- 2 points if no female characters were victimized, stereotyped or sexualized
- And 1 point if a sex scene shows foreplay before consummation, or if the female characters initiate or reciprocate sexual advances

Rees Davies Test -- a movie passes if...

- Every department has two or more women

The Waithe Test -- a movie passes if...

- There's a black woman in the work
- Who's in a position of power
- And she's in a healthy relationship

Methods

To conduct a more thorough and comprehensive analysis on gender imbalances in the current film industry, our group decided to utilize and combine four predefined tests — the Peirce Test, Feldman Score Test, Rees Davies Test, and Waithe Test — to form our own definition of a feminist movie. **In our definition, a movie is feminist if and only if it passes the Peirce Test, the Feldman Score Test, the Rees Davies Test, and the Waithe Test.** In the Movie class, we have defined a method called isFeminist() which helps us determine whether a Movie object is feminist according to our definition. Within isFeminist(), we first retrieved all the test results of the Movie object by calling getAllTestResults() and then we stored the results into a vector of String called testResults. By retrieving the test results at indices 1, 3, 8, 12, (incides at which the test results of the four tests are stored), we could tell if the Movie object passed the individual feminist tests by checking if the string is equal to "0." To ensure that isFeminist() would work for a movie that passed all four tests, in the main method of the Movie class we created a new Movie object called "Our Feminist Movie: Lady Birds," setting its test results to all zeros, and tested isFeminist() on this object we created.

In our MovieCollection class, we were able to find all the feminist movies in the provided dataset by calling findAllFeministMovies(). findAllFeministMovies() takes no parameters and returns a linked list, which is a specific type of collection of objects, of movies that passed all four tests. Within the method, we first created an empty linked list called feministMovies, which is used to store feminist movies. Then we looped over every single Movie object in the allMovies linked list to check if the movie object passed all four tests. To do so, we used a scanner to read through the test results of the four tests at their specific indices. Once a movie passes all four tests, we add the Movie object to the linked list, or collection of feminist movies, feministMovies. This method returns the linked list after the for loop is executed completely, and if there are no movies that passed all four tests, an error message will be printed out to indicate that there were no feminist movies in the given dataset. To ensure that this method would work for a movie that passed all four tests, we defined a helper method called addOurFeministMovie() where we created our own Movie object called "Our Feminist Movie: Lady Birds." We set its test results to "0"s for all the tests and added the Movie object to the allMovies linked list. In our testing for findAllFeministMovies() in the main method, "Our Feminist Movie: Lady Birds" was the only movie that passed all four tests.

If our available dataset contained more movies, this method would not need to be changed. allMovies, a linked list of Movie objects populated in the constructor that is essentially a list of all of the movies in our dataset, would get bigger due to the increase of movies. This, in turn, means that the number of iterations for the for loop would also get bigger. We would use the same conditional to check the separate strings at specific indices to determine if a movie was feminist or not. So, even though it might take longer to get the result of all the feminist movies, we would not need to alter the method.

Conclusions

While there was a smattering of movies that passed each individual test, as well as a few combinations of tests, the only movie that passed *our* feminist test (aka passed the Peirce Test, the Feldman Score Test, the Rees Davies Test, and the Waithe Test) was our ultra-feminist movie, "Our Feminist Movie: Lady Birds." Unfortunately, this is not surprising. The current film industry is plagued with misogyny and general inequity, even in the wake of the #MeToo Movement. And while there are directors, producers, actors, etc., many of whom are female, who are actively working towards creating a more equitable film industry, we aren't quite there yet. There is still a lack of POC in the film industry, and especially female-identifying POC. There is also a significant, but also unfortunately unsurprising, lack of indiginous, latinx, LGBTQ+ and disabled representation throughout this industry. In order for a movie to be truly feminist, women of all backgrounds, genders and abilities should be given equal opportunity to showcase their talents and expertise.

A possible new Bechdel test to add to the existing twelve is one that is more inclusive of non-binary gender representation and the inclusion of the LGBTQ+ community. All of the existing tests primarily focus on the representation of cis females, and seem to overlook the transgender and non-binary commmunity of actors. HOpefully, this would make the film and TV industry more aware of the lack of LGBTQ+ folks, which would then hopefully translate into it becoming more diverse and inclusive in all matters.

Code

Below you will find our implementations of the Actor, Movie, and MovieCollection classes with their respective documentation as well as author and version information, and our main methods for testing:

```
/**
 * Represents an object of type Actor. An Actor has a name and a gender.
 *
 * @author yl102, vs2, cl103
 * @version 12/10/2021
 */
public class Actor {
    private String name;
    private String gender;

    /**
     * Constructor for Actor.
     *
     * @param name The name of the actor
```

```
     * @param gender The gender of the actor
     */
    public Actor(String name, String gender){
        this.name = name.replace("\"", ""); // add name while removing double
quotes
        this.gender = gender.replace("\"", ""); // add gender while removing double
quotes
    }

    /**
     * This method is defined here because Actor (mutable) is used as a key in a
Hashtable.
     * It makes sure that the same Actors always have the same hash code.
     * So, the hash code of any object that is used as key in a hash table,
     * has to be produced on an *immutable* quantity,
     * like a String (such a string is the name of the actor in our case)
     *
     * @return an integer, which is the has code for the name of the actor
     */
    public int hashCode() {
        return name.hashCode();
    }

    /**
     * Returns the gender of this actor
     * @return gender of Actor
     */
    public String getGender(){
        return gender;
    }

    /**
     * Returns the name of this actor
     * @return name of Actor
     */
    public String getName(){
        return name;
    }

    /**
     * Sets the gender of this actor
     * @param g new gender of actor
     */
    public void setGender(String g){
        gender = g;
    }
```

```
    /**
     * Sets the name of this actor
     * @param n new name of actor
     */
    public void setName(String n){
        name = n;
    }

    /**
     * Returns a string representation of this Actor.
     * @return s the string representation
     */
    public String toString(){
        return ("Name: " + name + "| Gender: " + gender);
    }

    /**
     * Tests this actor against the input one and determines whether they are
equal.
     * Two actors are considered equal if they have the same name and gender.
     *
     * @return true if both objects are of type Actor,
     * and have the same name and gender, false in any other case.
     */
    public boolean equals(Object other) {
        if (other instanceof Actor) {
            return this.name.equals(((Actor) other).name) &&
            this.gender.equals(((Actor) other).gender); // Need explicit (Actor)
cast to use .name
        } else {
            return false;
        }
    }

    /**
     * Main method for testing
     */
    public static void main (String[] args){
        Actor a = new Actor ("Brandon", "Male");
        System.out.println("Testing getName() and getGender() for Brandon, Male.");
        System.out.println(a.getName());
        System.out.println(a.getGender());
        System.out.println();

        System.out.println("Testing setName() and setGender(), result should be
Kyle, Non-Binary.");
        a.setName("Kyle");
```

```
        System.out.println(a.getName());
        a.setGender("Non-binary");
        System.out.println(a.getGender());
        System.out.println();

        System.out.println("Testing toString() of Kyle, Non-Binary.");
        System.out.println(a);

        System.out.println();
        System.out.println("Creating new Actor: Miya, Female. Testing if Kyle
equals to Miya, Should return false");
        Actor b = new Actor ("Miya", "Female");
        System.out.println(a.equals(b));

        System.out.println();
        System.out.println("Testing hashCode() on Kyle.");
        System.out.println(a.hashCode());
    }

}
```

```
import java.util.*;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.File;
/**
 * Represents an object of type Movie.
 * A Movie object has a title, some Actors, and results for the twelve Bechdel
tests.
 *
 * @author yl102, vs2, cl103
 * @version 12/10/2021
 */
public class Movie {
    private String title;
    private Hashtable<Actor, String> actorsList;
    private Vector<String> result;

    /**
     * Constructor
     * Constructs a Movie object with a given title
     */
    public Movie(String title){
        this.title = title;
        actorsList = new Hashtable<Actor,String>();
```

```
    }

    /**
     * Returns the movie's title
     *
     * @return title of the movie
     */
    public String getTitle(){
        return title;
    }

    /**
     * populates the testResults vector with "0" and "1"s,
     * each representing the result of the corresponding test on this movie.
     *
     * @param results string consisting of 0's and 1's. Each one of these values
denotes the result of the corresponding test on this movie
     */
    public void setTestResults(String results){
        String [] array = results.split("[,]", 0); // create an array that is
populated with the contents of results, splitting at each ","
        result = new Vector<String>(Arrays.asList(array)); // turn the String array
into a vector of strings of the test results
    }

    /**
     * returns a Vector with all the Bechdel test results for this movie
     *
     * @return A Vector with the Bechdel test results for this movie:
     * A test result can be "1" or "0" indicating that this move passed or did not
pass the corresponding test.
     */
    public Vector<String> getAllTestResults(){
        return result;
    }

    /**
     * Reads the input file ("nextBechdel_castGender.txt"), and adds all its Actors
to this movie.
     * Each line in the movie has the following formatting:
     * Input String has the following formatting: "MOVIE
TITLE","ACTOR","CHARACTER_NAME","TYPE","BILLING","GENDER" Example of input:
"Trolls","Ricky Dillon","Aspen Heitz","Supporting","18","Male"
     *
     * @param actorsFile The data file to be read
     */
    public void addAllActors(String actorsFile){
```

```
        try{
            Scanner reader = new Scanner(new File(actorsFile));
            reader.nextLine(); // skip the first informational line
            reader.useDelimiter(",|\n");
            while (reader.hasNext()){
                if (reader.next().replace("\"","").equals(title)) { // if we are at
the correct movie in the data file
                    String name = reader.next(); // assign entries to appropriate
variables
                    reader.next();
                    String roleType = reader.next();
                    reader.next();
                    String gender = reader.next();

                    Actor a = new Actor(name, gender);
                    actorsList.put(a,roleType); // populate actorsList
                }
            }
            reader.close();
        }
        catch(IOException ex){
            System.out.println("File can't be read from. Operation addAllActors()
failed.");
        }
    }

    /**
     * Takes in a String, formatted as lines are in the input file
("nextBechdel_castGender.txt"),
     * generates an Actor, and adds the object to the actors of this movie.\
     *
     * @param line A string representation of the actor.
     * @return The Actor Object being added
     */
    public Actor addOneActor(String line){
        String [] array = line.split("[,]", 0); // turn line into an array of
strings splitting at each comma
        Actor b = new Actor (array[1], array[5]); // create Actor using information
from input line (name and gender)
        if (!actorsList.contains(b)){ // if the actor isn't in the hashtable
already
            actorsList.put(b,array[3].replace("\"", "")); // put actor in list with
key=Actor Object and value = role type
        }
        else{
            System.out.println(b.getName() + " is already in the hashtable");
        }
```

```
        return b;
    }

    /**
     * Returns the movie's actors in a Hashtable
     *
     * @return Hashtable of movie's actors
     */
    public Hashtable<Actor, String> getAllActors(){
        return actorsList;
    }

    /**
     *     returns a Linked List with all the actor names who played in this movie.
     *
     *     @return Linked List containing actor names
     */
    public LinkedList<String> getActors(){
        Enumeration<Actor> actorKeys = actorsList.keys(); // create an enumeration
containing the actors in actorsList
        LinkedList<String> actorNames = new LinkedList<String>();
        while (actorKeys.hasMoreElements()){
            Actor currentActor = actorKeys.nextElement();
            actorNames.add(currentActor.getName()); // add the name of the current
actor to actorNames
        }
        return actorNames;
    }

    /**
     * Tests this movie object with the input one and determines whether they are
equal.
     *
     * @param other The Movie Object to be compared
     * @return true if both objects are movies and have the same title,
     * false in any other case.
     */
    public boolean equals(Object other) {
        if (other instanceof Movie) {
            return this.title.equals(((Movie) other).title); // Need explicit
(Movie) cast to use .title
        } else {
            return false;
        }
    }

    /**
```

```
     * Returns a string representation of this movie.
     *
     * @return a string representation of movie
     */
    public String toString(){
        String s = "The movie title is: " + title + "\nActors in the movie are:
\n"+actorsList.toString();
        s+= "\nThe test results for this movie are: " + getAllTestResults() + "\n";
        return s;
    }

    /**
     * Decides if this movie is feminist.
     * It is considered feminist if it passes the Pierce Test, Feldman Score Test,
     * Rees Davies Test, and Waithe Test.
     *
     * @return True if the movie is feminist. False if not.
     */
    public boolean isFeminist(){
        Vector<String> testResults = getAllTestResults();

        //if passes Pierce Test, Feldman Score Test, Rees Davies Test, and Waithe
Test, it isFeminist() = true
        if ((testResults.get(1).equals("0")) && (testResults.get(3).equals("0")) &&
(testResults.get(8).equals("0")) && (testResults.get(12).equals("0"))){
            return true;
        }
        return false;
    }

    /**
     * Main method for testing
     */
    public static void main (String [] args){
        Movie a = new Movie("Alpha");
        System.out.println("Testing getTitle() method on movie named Alpha, should
return Alpha.");
        System.out.println(a.getTitle());

        System.out.println("\nTesting addAllActors() method on new movie named
Alpha. (There are no test results added for Alpha yet)");
        a.addAllActors("data/small_castGender.txt");
        System.out.println(a);

        System.out.println("\nTesting addOneActor() method on new movie named
Gamma. Should return \"Name: Cassi Davis| Gender: Female\"");
        Movie b = new Movie ("Gamma");
```

```
        System.out.println(b.addOneActor("\"Gamma\",\"Cassi Davis\",\"Aunt
Bam\",\"Supporting\",\"2\",\"Female\""));

        System.out.println("\nTesting getActors() method on movie named Gamma.
Should return \"Cassie Davis\"");
        System.out.println(b.getActors());

        System.out.println("\nTesting getAllActors() method on movie named Gamma.
Should return \"Name: Cassi Davis| Gender: Female=Supporting\"");
        System.out.println(b.getAllActors());

        System.out.println("\nTesting setTestResults() method. Should print
\"0,0,0,1,0,0,0,1,0,0,1,1,1\"");
        b.setTestResults("0,0,0,1,0,0,0,1,0,0,1,1,1");
        System.out.println(b);

        System.out.println("\nTesting getAllTestResults() method. Should return
\"0,0,0,1,0,0,0,1,0,0,1,1,1\"");
        System.out.println(b.getAllTestResults());

        System.out.println("\nTesting equals() method on Alpha and Gamma movies.
Should return false.");
        System.out.println(a.equals(b));

        System.out.println("\nTesting equals() method on if Alpha is equal to
itself. Should return true.");
        System.out.println(a.equals(a));

        a.setTestResults("0,1,0,1,0,0,0,0,1,0,0,0,1");
        System.out.println("\nTesting isFeminist() method on Alpha. Should return
false");
        System.out.println(a.isFeminist());

        System.out.println("\nTesting isFeminist() method on Gamma. Should return
false");
        System.out.println(b.isFeminist());

        Movie beta = new Movie("Beta");
        System.out.println("\nTesting addAllActors for new movie, Beta (There are
no test results for Beta yet)");
        beta.addAllActors("data/small_castGender.txt");
        System.out.println(beta);


System.out.println("-----------------------------------------------------------")
;
        System.out.println("Testing movie \"Hidden Figures\" from
```

```
nextBechdel_castGender.txt file.");

        Movie c = new Movie("Hidden Figures");
        System.out.println("Testing getTitle() method on movie named Hidden
Figures, should return Hidden Figures.");
        System.out.println(c.getTitle());

        System.out.println("\nTesting setTestResults() method on movie named Hidden
Figures to 0,0,0,0,0,1,0,1,0,1,1,1,1");
        c.setTestResults("0,0,0,0,0,1,0,1,0,1,1,1,1");

        System.out.println("\nTesting addAllActors() method on new movie named
Hidden Figures.");
        c.addAllActors("data/nextBechdel_castGender.txt");
        System.out.println(c);

        System.out.println("\nTesting getActors() method on movie named Hidden
Figures. Should return LinkedList of actors names.");
        System.out.println(c.getActors());

        System.out.println("\nTesting getAllActors() method on movie named Hidden
Figures. Should return HashTable of actor objects");
        System.out.println(c.getAllActors());


        System.out.println("\nTesting isFeminist() method on Hidden Figures. Should
return false");
        System.out.println(b.isFeminist());


System.out.println("----------------------------------------------------------------")
;

        Movie d = new Movie("Our Feminist Movie: Lady Birds");
        System.out.println("Testing getTitle() method on movie named Our Feminist
Movie: Lady Birds, which we created. Should return Our Feminist Movie: Lady
Birds.");
        System.out.println(d.getTitle());

        System.out.println("\nTesting addOneActors() method on new movie named Our
Feminist Movie: Lady Birds.");
        d.addOneActor("\"Our Feminist Movie: Lady Birds\",\"Crystal Luo\",\"the
lady\",\"Lead Ensemble Member\",\"1\",\"Female\"");
        System.out.println(d);

        System.out.println("\nTesting getActors() method on movie named Our
Feminist Movie: Lady Birds. Should return LinkedList of actors names.");
```

```
        System.out.println(d.getActors());

        System.out.println("\nTesting getAllActors() method on movie named Our
Feminist Movie: Lady Birds. Should return HashTable of actor objects");
        System.out.println(d.getAllActors());

        System.out.println("\nSetting test results for Our Feminist Movie: Lady
Birds to 0,0,0,0,0,0,0,0,0,0,0,0,0");
        d.setTestResults("0,0,0,0,0,0,0,0,0,0,0,0,0");

        System.out.println("This is Our Feminist Movie:\n" + d);

        System.out.println("\nTesting isFeminist() method on Our Feminist Movie:
Lady Birds. Should return true");
        System.out.println(d.isFeminist());
    }

}
```

```
import java.util.LinkedList;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.File;
import java.util.Scanner;

/**
 * Represents a collection of Movies.
 * Uses a LinkedList to hold the movie objects.
 * Movie data (title and test results) are coming from a file named
"nextBechdel_allTests.txt".
 * Data regarding actors who participated in each movie is read from a file named
"nextBechdel_castGender.txt". Both files are provided in the "data" folder.
 *
 * @author yl102, vs2, cl103
 * @version 12/10/2021
 */

public class MovieCollection{
    private LinkedList<Movie> allMovies;
    private LinkedList<Actor> allActors;
    private String testsFileName;
    private String castsFileName;
    private String testResults;

    /**
```

```
     * Constructor for objects of class MovieCollection
     */
    public MovieCollection(String testsFileName, String castsFileName){
        allMovies = new LinkedList<Movie>();
        allActors = new LinkedList<Actor>();
        this.testsFileName = testsFileName;
        this.castsFileName = castsFileName;
        readMovies(); // add in the movies to allMovies
        readCasts(); // add in actors to allActors
    }

    /**
     * Returns all the movies in a LinkedList
     *
     * @return Linked List of Movie Objects
     */
    public LinkedList<Movie> getMovies(){
        return allMovies;
    }

    /**
     * Returns the titles of all movies in the collection
     *
     * @return Linked List of titles of movies in collection
     */
    public LinkedList<String> getMovieTitles(){
        LinkedList<String> allMovieTitles = new LinkedList<String>();
        for (int i = 0; i<allMovies.size(); i++){ // iterate through allMovies and
add titles to linked list
            allMovieTitles.add(allMovies.get(i).getTitle()); //
        }
        return allMovieTitles;
    }

    /**
     * Returns all the Actors in the collection
     *
     * @return Linked List of Actor Objects
     */
    public LinkedList<Actor> getActors(){
        return allActors;
    }

    /**
     *     Returns the names of all actors in the collection
     *
     *     @return Linked List of actor names
```

```
     */
    public LinkedList<String> getActorNames(){
        LinkedList<String> allActorNames = new LinkedList<String>();
        for (int i = 0; i<allActors.size(); i++){ // iterate through allActors and
add names to linked list
            allActorNames.add(allActors.get(i).getName());
        }
        return allActorNames;
    }

    /**
     * Reads the input file, and uses its first column (movie title) to create all
movie objects.
     */
    private void readMovies(){
        try{
            Scanner reader = new Scanner(new File(testsFileName));
            reader.nextLine(); // skip the first informational line
            reader.useDelimiter(",|\n"); // set delimiters
            while (reader.hasNext()){
                String title = reader.next(); // the first entry is the title of
the movie
                if (!allMovies.contains(title)){ // if the movie isn't already in
allMovies...
                    Movie a = new Movie(title);
                    testResults = "";
                    for (int i = 0; i<13; i++){ // go through the next 13 entries
of test results and record them in testResults
                        testResults += reader.next() + ",";

                    }
                    //System.out.println();
                    a.setTestResults(testResults); // attach test results to Movie
object
                    a.addAllActors(castsFileName); // attach Actors to Movie object
                    allMovies.add(a); // add new Movie object to allMovies
                }
            }
            reader.close();
        } catch(IOException ex){
            System.out.println("File can't be read from. Operation readMovies()
failed.");
        }
    }

    /**
     * Reads the casts for each movie, from input casts file;
```

```
     * Assume lines in this file are formatted as follows:
     * "MOVIE","ACTOR","CHARACTER_NAME","TYPE","BILLING","GENDER"
     * For example: "Trolls","Ricky Dillon","Aspen Heitz","Supporting","18","Male".
     */
    private void readCasts(){
        try{
            Scanner reader = new Scanner(new File(castsFileName));
            reader.nextLine(); // skip the first informational line
            reader.useDelimiter(",|\n");
            while (reader.hasNext()){
                String title = reader.next(); // added new variables for each data
info debugging sake so it would be clear...
                String name = reader.next();
                String characterName = reader.next();
                String type = reader.next();
                String billing = reader.next();
                String gender = reader.next();
                Actor a = new Actor(name, gender);
                allActors.add(a); // add the Actor to allActors
            }
            reader.close();
        }catch(IOException ex){
            System.out.println("File can't be read from. Operation readCasts()
failed.");
        }
    }

    /**
     * Adds Our Feminist Movie: Lady Birds to the movie collection.
     * This is for testing purposes since Our Feminist Movie: Lady Birds is
ulta-feminist and passes all tests.
     *
     * @return A linked list of all movies in the collection, including Our
Feminist Movie: Lady Birds
     */
    public LinkedList<Movie> addOurFeministMovie(){
        // adding a very feminist movie that will pass all tests :)
        Movie ourFeministMovie = new Movie("Our Feminist Movie: Lady Birds");
        ourFeministMovie.setTestResults("0,0,0,0,0,0,0,0,0,0,0,0,0");
        ourFeministMovie.addOneActor("\"Our Feminist Movie: Lady Birds\",\"Crystal
Luo\",\"the lady\",\"Lead Ensemble Member\",\"1\",\"Female\"");
        ourFeministMovie.addOneActor("\"Our Feminist Movie: Lady Birds\",\"Veronica
Lin\",\"Saoirse Ronan\",\"Lead Ensemble Member\",\"2\",\"Female\"");
        ourFeministMovie.addOneActor("\"Our Feminist Movie: Lady Birds\",\"Tori
Stiegman\",\"the bird\",\"Lead Ensemble Member\",\"3\",\"Female\"");
        allMovies.add(ourFeministMovie);
```

```
        return allMovies;
    }

    /**
     * Returns a list of all Movies that pass the n-th Bechdel test
     *
     * @param n The number of the Bechdel test of interest
     * @return A Linked List of all the movies that pass
     */
    public LinkedList<Movie> findAllMoviesPassedTestNum(int n){
        LinkedList<Movie> passedMovies = new LinkedList<Movie>();
        for (int i = 0; i < allMovies.size(); i ++){ // iterate through allMovies
            Movie currentMovie = allMovies.get(i);
            if (currentMovie.getAllTestResults().get(n - 1).equals("0")){ // if the
current movie passed the test in question
                passedMovies.add(currentMovie); // add the current movie to
passedMovies list
            }
        }

        if (passedMovies.size() == 0){ // if there are no movies that passed the
test in question...
            System.out.println("There are no movies that passed " + n + ".");
        }
        return passedMovies;
    }

    /**
     * Returns a list of all movies that passed either the x or y Bechdel test
     *
     * @param x the number of the Bechdel test of interest
     * @param y the other number of the Bechdel test of interest
     * @return A linked list of all the movies that passed either x or y test
     */
    public LinkedList<Movie> findAllMoviesPassedEither(int x, int y){
        LinkedList<Movie> passedMovies = new LinkedList<Movie>();
        for (int i = 0; i < allMovies.size(); i ++){ // iterate through allMovies
            Movie currentMovie = allMovies.get(i);

            // if the current movie passed EITHER of the tests in question
            if (currentMovie.getAllTestResults().get(x - 1).equals("0") ||
(currentMovie.getAllTestResults().get(y - 1).equals("0"))){
                passedMovies.add(currentMovie);
            }
        }
        if (passedMovies.size() == 0){ // if there are no movies that passed the
tests in question...
```

```
            System.out.println("There are no movies that passed either " + x + " or
" + y + ".");
        }
        return passedMovies;
    }

    /**
     * Returns a list of all movies that passed one test but not the other test
     *
     * @param x the number of the Bechdel test of interest
     * @param y the other number of the Bechdel test that is not passed
     * @return A linked list of all the movies that passed x but not y test
     */
    public LinkedList<Movie> findAllMoviesPassedOneNotOther(int x, int y){
        LinkedList<Movie> passedMovies = new LinkedList<Movie>();
        for (int i = 0; i < allMovies.size(); i ++){ // iterate through allMovies
            Movie currentMovie = allMovies.get(i);

            // if the current movie passed test x and not test y...
            if (currentMovie.getAllTestResults().get(x - 1).equals("0") &&
(currentMovie.getAllTestResults().get(y - 1).equals("1"))){
                passedMovies.add(currentMovie);
            }
        }
        if (passedMovies.size() == 0){ // if there are no movies that passed x and
did not pass y...
            System.out.println("There are no movies that passed " + x + " and did
not pass " + y + ".");
        }
        return passedMovies;
    }

    public LinkedList<Movie> findAllFeministMovies() {
        LinkedList<Movie> passedMovies = new LinkedList<Movie>();
        for (int i = 0; i < allMovies.size(); i ++){ // iterate through allMovies
            Movie currentMovie = allMovies.get(i);

            // if the current movie passed Pierce Test, Feldman Score Test, Rees
Davies Test, and Waithe Test, then isFeminist() = true...
            if (currentMovie.isFeminist()){
                passedMovies.add(currentMovie);
            }
        }
        if (passedMovies.size() == 0){ // if there are no movies that passed
isFeminist()...
            System.out.println("There are no feminist movies.");
        }
```

```
        return passedMovies;
    }

    /**
     * Main Method for testing
     */
    public static void main(String[] args){
        System.out.println("***Testing MovieCollection on \"small_allTests.txt\"
and \"small_castGender.txt.\"***");
        MovieCollection a = new MovieCollection("data/small_allTests.txt",
"data/small_castGender.txt");

        System.out.println("\nTesting getMovies(). Should return LinkedList of
Movie Objects.");
        System.out.println(a.getMovies());

        System.out.println("\nTesting getMovieTitles(). Should return LinkedList of
Movie Titles.");
        System.out.println(a.getMovieTitles());
        System.out.println("\nTesting getActors(). Should return LinkedList of
Actor Objects.");
        System.out.println(a.getActors());
        System.out.println("\nTesting getActorNames(). Should return LinkedList of
Actor Names.");
        System.out.println(a.getActorNames());

        System.out.println();
        System.out.println("\n***EXTENSIVE TESTING***");
        System.out.println("Find all the movies that passed the Bechdel Test:");
        System.out.println(a.findAllMoviesPassedTestNum(1));

        System.out.println();
        System.out.println("Find all the movies that passed the Pierce or the
Landau Test:");
        System.out.println(a.findAllMoviesPassedEither(2, 3));

        System.out.println();
        System.out.println("Find all the movies that passed the White but not the
Rees-Davis Test, should print out the error message.");
        System.out.println(a.findAllMoviesPassedOneNotOther(12, 13));

        System.out.println("Find all the movies that passed the Feldman Test,
should print out the error message.");
        System.out.println(a.findAllMoviesPassedTestNum(4));

        System.out.println();
        System.out.println("Find all the movies that passed the Feldman but not the
```

```
Villareal Test, should print out the error message.");
        System.out.println(a.findAllMoviesPassedOneNotOther(4, 5));
        System.out.println();


System.out.println("-----------------------------------------------------------------
-------------------------");
        System.out.println("***Testing MovieCollection on
\"nextBechdel_allTests.txt\" and \"nextBechdel_castGender.txt.\"***");

        MovieCollection b = new MovieCollection("data/nextBechdel_allTests.txt",
"data/nextBechdel_castGender.txt");
        System.out.println("Let's add Our Feminist Movie: Lady Birds to the data
set...");
        b.addOurFeministMovie(); // adding our feminist movie to the data set...

        // System.out.println("*****Testing getMovies(). Should return LinkedList
of Movie Objects.*****");
        // System.out.println(b.getMovies());

        // System.out.println("*****Testing getMovieTitles(). Should return
LinkedList of Movie Titles.*****");
        // System.out.println(b.getMovieTitles());
        // System.out.println("*****Testing getActors(). Should return LinkedList
of Actor Objects.*****");
        // System.out.println(b.getActors());
        // System.out.println("*****Testing getActorNames(). Should return
LinkedList of Actor Names.*****");
        // System.out.println(b.getActorNames());

        // System.out.println();
        // System.out.println("***EXTENSIVE TESTING***");
        // System.out.println("*****Find all the movies that passed the Bechdel
Test.*****");
        // System.out.println(b.findAllMoviesPassedTestNum(1));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed the Pierce or
the Landau Test.*****");
        // System.out.println(b.findAllMoviesPassedEither(2, 3));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed the White but
not the Rees-Davis Test.*****");
        // System.out.println(b.findAllMoviesPassedOneNotOther(12, 13));

        // System.out.println();
```

```
        // System.out.println("*****Find all the movies that passed the Bechdel but
not the Pierce Test.*****");
        // System.out.println(b.findAllMoviesPassedOneNotOther(1, 2));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed the Feldman
Test.*****");
        // System.out.println(b.findAllMoviesPassedTestNum(4));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed the White or
the Rees-Davis Test.*****");
        // System.out.println(b.findAllMoviesPassedEither(12, 13));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed the Feldman but
not the Villareal Test.*****");
        // System.out.println(b.findAllMoviesPassedOneNotOther(4, 5));

        // System.out.println();
        // System.out.println("*****Find all the movies that passed OUR feminist
test.*****");
        // System.out.println(b.findAllFeministMovies());
    }
}
```

Collaboration

Overall, our team worked incredibly well together. We met up early for an initial meeting and came up with a plan of attack that each of us felt comfortable with and that split up the work evenly. We met as a group multiple times throughout the course of this project both in person and on Zoom, which is where the majority of the coding and debugging took place. While in person, we worked on one person's laptop, occasionally switching off drivers. While one person was typing, the other two would look on (either on their own computer on Zoom while the code was shared or by simply looking on to the working laptop), offering opinions and coming up with possible ways to code whatever we were working on.

We also chatted about the project in a group chat. This is where, if one or two of us were working on the project in some way (going through the data files, writing javadoc or inline comments, working on debugging a certain method if they had a random spark of inspiration, etc), we would share what we were doing with the group. Specifically, we would send screenshots and detailed descriptions of the work we were doing so the others could see exactly what was going on, and offer input or celebrate when things actually started to work!

To finish the report, we took a slightly different approach and decided to divide-and-conquer. We divided up the sections amongst each other to make sure that we all had about the same amount of work to do.