Movie Search

Team 4

Michael Totaro Unity ID: mtotaro , Ryan Morgan Unity ID: rpmorgan , Logan Williams Unity ID: lrwilli7

NOTE: Your final documents should **NOT** contain any << >>'s or any comments or suggestions given for each section.

Introduction

Our team will create a movie searching application that allows a user to search through a list of movies using several different search functions. The user can search by the name of the movie, the year it was released, their desired length of movie, or by different genres. The program will be started on the command line and the user must provide a properly formatted text file which contains the list of movies in which they want to search. The program will then open a GUI that contains a menu with the different search options. The GUI will display the search results as a list of movies matching the requested criteria. The user can search for movies indefinitely until the program is quit.

Software Requirements

MovieSearch allows a user to search through a list of movies. The user starts the program on the command line and includes an input text file which contains the list. The program opens up a GUI with which the user can search through the full list based on different information about the movies they want to find.

User Interface

The user starts the program on the command line and must include exactly one command line argument, which is the name of the text file that contains the list of movies. For example,

```
> java MovieSearch movies.txt
```

If the command line argument is input correctly, and the input file is properly formatted, then the program will start a GUI. The GUI will contain several different options to search through the list of movies. When an option is selected, the GUI will immediately display the search results. The GUI will stay open until the user selects the quit option.

Input File

The input text file containing the list of movies must be properly formatted for MovieSearch to work. The list must be organized into rows of movies which are made up of 4 tab-delimited columns with the following information in this specific order:

- Title (may contain spaces)
- Year
- Length (minutes)
- Genre (one or more of the following genres, separated by commas, but no spaces)
    - Action
    - Adventure
    - Animation
    - Biography
    - Comedy
    - Crime
    - Documentary
    - Drama
    - Family
    - Fantasy
    - History
    - Horror
    - Musical
    - Mystery
    - Romance
    - Sci-Fi
    - Sport
    - Thriller
    - War
    - Western

For example, one line of the file should look like:

```
Jaws    1975    124     Adventure,Drama,Thriller
```

The first line of the text file must be a header record with column names, and not contain any specific movie, as this line will be skipped by the program.

Search Options

The program will display five options for the user to select.

- All
    - This option displays the entire list of movies.
- Title
    - This option displays all movies which contain the specified search term entered by the user. The user can enter one or more words, but the program only finds movies which contain the words in the exact way they were entered. For example, searching “Lord of the Rings” does not return results that contain “Lord” or “Rings” in the title; it only returns titles that contain “Lord of the Rings”.
- Year
    - This option displays all movies which were released in the entered year, between 1890 - 2020.
- Length
    - This option displays all movies shorter than, equal to, or longer than a specific length in minutes. The option and length are entered by the user.
- Genre
    - This option displays all movies matching the genre(s) selected by the user.

Error Handling

When starting the program, if there is not exactly one command line argument, the program outputs a usage message and quits. For example,

```
> java MovieSearch
Usage: java MovieSearch infile
$
```

If the input file does not exist, the program outputs an error message and quits. For example,

```
> java MovieSearch notmovies.txt
File not found
$
```

When searching for movies by year, if the user enters a year outside of the valid range (1890 - 2020), the program displays an error message: “Invald year. Please try again.”

When searching for movies by length, if the user enters a length less than or equal to zero, the program displays an error message: “Invalid length. Please try again.”

When searching for movies by genre, if the user enters an invalid genre, the program displays an error message: “Invalid genre. Please try again.”

Software Design

Class Descriptions

- in front of a name means private
+ in front of a name means public
~ in front of a name means default

MovieGUI

- movieList: MovieList -> An array containing movie objects
- label: JLabel -> Label for the label, "Please choose an option from the menu".
- displayMovies: JButton -> The button for displaying all movies.
- displayString: JButton -> The button for searching movies by string.
- displayYear: JButton -> The button for searching movies by year.
- displayGenre: JButton -> The button for searching movies by genre
- displayLength: JButton -> The button for searching movies by runtime
- area: JTextArea -> The uneditable text area where results are displayed
- scrollPane: JScrollPane -> Scroll bar for area
+ SET_SIZE_WIDTH: int -> The width of the frame
+ SET_SIZE_HEIGHT: int -> The height of the frame
+ AREA_WIDTH: int -> Width of JTextArea “area”
+ AREA_HEIGHT: int -> Height of JTextArea “area”
+ MIN_YEAR: int -> Earliest year a movie could’ve been made in to be considered valid
+ MAX_YEAR: int -> Latest year a movie could’ve been made in to be considered valid

+ MovieGUI(MovieList) -> Constructor for the class. Initializes movieList parameter. Set’s up the frame. Set’s Sets title to "Movie Search GUI". Initializes the X value of size to SET_SIZE_WIDTH and the Y value to SET_SIZE_HEIGHT. Sets the default close operation to x being pressed. Creates a layout and sets flowlayout. Creates new labels and buttons with text matching their function. Sets foreground and background colors. Adds label, button, and scroll pane to container. Makes the the text area uneditable. Sets the frame to visible. Adds an action listener for each button.
+ actionPerformed(ActionEvent): void -> Creates different actions for each button being pressed. If the the button for displaying all movies is pressed, all movies will be

displayed. If the button for searching by string is pressed a dialog box will open and reprompt until you enter a string that returns valid results that contain the string. If you press the search by year button dialog box will open until you enter text that is all integers and between 1890 - 2020. Then will display results for movies released in the year that was searched for. If the display by genre button is pressed a dialog box will open with a search bar and the user will type in genres from a displayed menu. Once the user enters a valid genre, or genres, results will be displayed if they are the genre that was searched. If the display by length button is pressed, the user will be prompted to enter an integer, and it they type in something that is not an integer they will be prompted. Once they enter a valid integer another dialog box will open and the user will have to decide whether they want to display movies that are longer, equal to, or shorter in length to the given integer.

+ checkInt(String): boolean -> Accepts string as input and returns true if all characters in the string are integers. If they aren't it returns false.

//Accepts string as a parameter
//Checks if all characters in the string are integers
//If they are all integers method returns true
//If they are not all integers, the method returns false.
public static boolean checkInt(String str)

Movie

- title: String -> title of the movie object
- year: String -> year the movie object was released
- length: int -> Length in minutes of the movie object
- genre: String -> The genre of the movie object.

+ Movie(String, String, String, String) -> Constructor for movie object. Initializes the class's instance variables using the parameter values.
+ getTitle(): String -> Returns the title of the movie object
+ getYear(): String -> Returns the year the movie object was released in.
+ getLength(): int -> Returns the runtime length (in minutes) of the movie object.
+ getGenre(): String -> Returns the genre of the movie object
+ equals(Object): boolean -> Compares the passed object with the class's instance variables. If the objects are the same, returns true. If not, returns false.
+ toString(): String -> Returns the movie object as a string.

MovieList

- movieList: MovieList -> Array of movie objects
- movieCount: Int -> Number of movies in array

+ MovieList(): int -> Constructor for MovieList object. Initializes size of Array to the parameter "size" and sets the movieCount to 0.
+ addMovie(String, String, String, String): void -> Accepts a title, year, length, and genre as parameters and creates new movie objects using them. Then adds the movie object to the movieList and increments the size of the

movieList by 1.

+ addMovie(Movie): void -> Adds movie to MovieList object if Movie object already exists (search results list). The movie object will be passed as a parameter.

+ titleSearch(String): MovieList -> Accepts a string as a parameter then creates a new MovieList object called searchResults. An array named movieList contains all the movies in a given file. The movieList is then iterated through and for every element a new movie object is created. The getTitle() method is then called for the movie object and is assigned to a variable, then this variable is converted to all lowercase and if the title of the movie contains the passed string the movie will be added to the MovieList searchResults. When the loop terminates the searchResults MovieList will be returned.

+ yearSearch(int): MovieList -> Accepts an integer as a parameter then creates a new MovieList object called searchResults. An array named movieList contains all the movies in a given file. The movieList is then iterated through and for every element a new movie object is created. The getYear() method is then called for the movie object and is assigned to a variable, then this variable is converted to an integer using the Integer.parseInt() method. If the passed parameter equals the year the movie was released, the movie is add to the MovieList searchResults. When the loop terminates the searchResults MovieList will be returned.

+ lengthSearch(int, int): MovieList -> Accepts two integers as parameters one for the length searched by the user, the other for whether the user wants to find movies that are longer, equal to, or shorter in length. Then creates a new MovieList object called searchResults. An array named movieList contains all the movies in a given file. The movieList is then iterated through and for every element a new movie object is created. The getLength() method is then called for the new movie object is assigned to a variable. If the integer passed for whether the user wants to display movies longer, equal to, or shorter in length is 0 movies shorter than the given length will be added to the MovieList searchResults. If the number is 1 movies equal in length will to the passed parameter will be added to searchResults. If the number is 2 movies longer than the passed parameter will be added to searchResults. When the loop terminates the searchResults MovieList will be returned.

+ genreSearch(String[ ]): MovieList -> Accepts a string array as a parameter. Then creates a new MovieList object called searchResults. An array named movieList contains all the movies in a given file. The movieList is then iterated through and for Every element a new movie object is created. The getGenre method is then called on the movie and a new string variable is created for the movies genre. This variable is then converted to lowercase and the passed array of genres is iterated through and if the movies genre contains the genre in the array the movie is added to searchResults. When the loop terminates, searchResults is returned.

+ toString(): String -> Converts all elements in the movieList array to strings and then adds them to a main string. When the loop terminates the main string is returned.

<u>**MovieSearch**</u> *Not used to create objects

+ main(String[] args): void -> Assigns the command line argument to a string variable. Then calls a method to create a scanner for the input file. Then calls a method that uses the input scanner to create a MovieList object containing the entire list of movies. Creates a new MovieGUI object and passes the MovieList object as an argument
+ getInput(String): Scanner -> Accepts a string as a parameter, which is assigned to the command line argument. Then the method will try to create a new FileInputStream using the string. If a FileNotFoundException occurs a message will be displayed and the program will exit.
+ getMovieList(Scanner): MovieList -> Accepts a file as a parameter. Then creates a scanner for the file and uses the file scanner to create a size for the array. The first line is skipped, but the rest on the lines are used to create the size for the MovieList object. Then a new scanner is created for the file and the first line is skipped. Each line in the file is split and made into a movie object that is added to the MovieList. Once the while loop terminates the MovieList will be returned.

//assigns only command line argument (input file) to a string variable
//calls a method to create a scanner for the input file
//calls a method that uses the input scanner to create a MovieList object containing the entire list of movies
//creates a new MovieGUI object and passes the MovieList object as an argument
//Accepts command line arguments as parameter
//Doesn’t return anything
public static void main(String[] args)

//creates scanner for text file, by accepting an input file and creating an FileOutputStream for that file
//Accepts file as parameter
//If FileNotFoundException error occurs prints out "File not found" and exits the program
//Returns file scanner
public static Scanner getInput(String file)

// accepts file as parameter
//Make a new scanner called inputForSize to read the file
//Skip first line of file
//Make a variable called size that iterates by one for every line in the file
// create new empty MovieList object the size of the “size” variable.\
//Create another new scanner called inputForMovies to read the file
// skip header line
// Reads each line in the file and creates a string for each line.
// Splits line string by tab delimiters then makes an array of the strings.
// Then passes strings as arguments to addMovie method, adding each movie to the MovieList
//Returns movielist
public static MovieList getMovieList(Scanner input)

Implementation

We plan to use a variety of concepts from the course to implement our software.

•For our movie program we used ideas of interacting classes, constructors, “return” keywords , boolean method, and “this” keywords.
•The constructor movie will accept various parameters and then use the “this” keyword to refer to the parameters in the class.
•The return will be used to return certain parameters when the methods are called in different classes.
•We will use the boolean class to accept and object and use the “instanceof” keyword to check whether it’s a movie or not. If it is, the different parts of it are compared with the current object and return true if they’re the same, if not false.

•In the MovieGUI we used java.awt.* and java.awt.event.* for the actionListener so that when a button is pressed the GUI responds.
•The ActionListener extension is used so that the GUI can interact with a user.
•We used private classes so that the areas can be used in the class and not overridden.
•We used various GUI features such as Jbuttons, so that the user can easily choose between the options.
•We used JTextAreas to display text. We used a JScrollPane so that the user can scroll through the results that are being displayed in the textarea. We used JLabel to label certain things to signify what to do to the user, such as to choose an option from the menu.
•We used class constants so that the user can easily understand what certain numbers mean and can be used between methods.
•We used a constructor to create the GUI display. Within the constructor we used setSize, setTitle, setLocation, setDefaultCloseOperation methods. The setSize method sets the length and width of the screen. The set location sets the location of the GUI. The setDefaultCloseOperation tells the program to exit when the x is clicked.
•The container is used to store various objects and the getContentPane is used to get all the objects to be displayed in the pane. The setLayout sets the container layout. The setBackground sets the color for the background. The set foreground sets the color for the foreground.
•The “add” keyword adds the components to the container.
•The setEditable(false) method makes it so you can’t type in or edit the text in the JTextArea. The setVisible(True) makes it so you can see the GUI.
•The addActionlistener makes the buttons functional.
•The e.getSource lets you press buttons and have them do certain actions.
•The area.append() method adds text to the text area. area.setText(“”) clears the text area of any previous text.
•The JOptionPane.showInputDialog() creates a dialog box with certain text.
•The toLowerCase() method turns all characters in a string to lowercase.
•Integer.ParseInt converts a string to an integer. The .equals() method is used to compare strings. The .replace() method is used to replace characters in a string.
•For loops are used to iterate through arrays and modify and check their elements.
•Arrays are used to hold text. The .contains() method is to check if a string contains another string.
•The .split() method is used to break apart a string at a certain character. The JOptionPane.YES_NO_CANCEL_OPTION method is used to display three options to the user. The JOptionPane.QUESTION_MESSAGE is used to ask the user a question and get input. The keyword “null” is used to check if a string is empty.
•NumberFormatException is used to check if a string contains characters that aren’t integers. The MovieList program passes the movie filename (or a Scanner to the movie file) to the constructor when creating a MovieList object.
•Then the constructor can populate the movie array with all the movie information from the file (maybe Movie[] movies). This object can then be used by the GUI to call the search methods in the actionPerformed method (displaying the results in the output area).

•For the MovieSearch program Scanners are used to scan files. The try and catch technique is used to check whether or not a file exists before assigning it to a variable.If it does it will be assigned to a variable name.
•FileNotFoundException is used to determine whether or not a file exists. The .nextLine() method is used to iterate through files.

Testing

You will need two files to perform these tests
•movies.txt
•shortmovielist.txt

There are no special testing scenario requirements for our project.

Team Reflection

During this project we faced my problems and learned many valuable lessons. Our team learned a lot about communication during this project, especially learning how to communicate over long distances. We also learned a lot about time management and how in order to meet deadlines you need to plan ahead and not procrastinate. We also learned a lot about the software development process and the planning, time, effort, and patience that is required to create good software.

In terms of the actual developmental process of our project we faced many issues. Sometimes we had difficulty deciding how to design our program, but we managed to discuss it and work it out as a team. Sometimes we had difficulty getting our program to work and we bounced ideas off of each other until we figured out a good idea. For example, we were trying to decide how the user should search results. We considered many options such as a search bar, dialog boxes, new buttons, etc. and decided that dialog boxes would work best for our vision. We also helped each other figure out how to fix bugs, such as while loops not breaking and text not being displayed properly.

Overall, we learned that in the end a huge part of developing good software depends much more on your team's ability to work together and effort, as opposed to being based solely on your programming abilities. We also learned that the software development process involves lots of challenges, and that just because you’re struggling at one point, does not mean that you should get discouraged or that you won’t eventually solve the issue. We also developed a better ability to use resources. Sometimes you can’t remember certain functions or coding techniques off the top of your head, and I feel like it helped us learn how to incorporate elements from past assignments into our current project, as well as learning how to go back over resources such as previous presentations, exercises, etc. to improve our project. This has been an extremely beneficial experience for all of us in terms of growth and we believe it will help us in our future programming endeavors.