Project 3

Computer Science 2334
Spring 2016

This project is group work. Group composition will be determined later this week. Each student should complete and submit the draft design individually.

User Request:

"Create a sortable and searchable data system for movies, broadcast series, and (some of) the people who make them, that uses text and binary input and output and a graphical data display."

Milestones:

1. Implement **Serializable** for the classes necessary to save and load all application data. *10 points*
2. Use object serialization to save and load the application data to and from a binary file. *15 points*
3. Implement a simple graphical display for showing counts of the application data. *25 points*
4. Create appropriate classes to store information on media makers. *10 points*
5. Use **LinkedHashMap**s to save to and retrieve information on media makers. *10 points*

► Develop and use a proper design. (See, in particular, Milestone 4, above.) *15 points*

► Use proper documentation and formatting. *15 points*

Description:

An important skill in software design is extending the work you have done previously. For this project you will rework Project 2, adding one or more classes for additional types of data, implementing object serialization for input and output, using the Java **LinkedHashMap** class, and adding a graphical display.

Besides the types of data described in Project 2, this project will include data of another type – data on some of the people who work to make movies and broadcast series. For this project, these *media makers* will include actors, directors, and producers, although in reality there are far more people involved in the making of media than just these.

Operation:

When MDb starts up, it will ask "Read (t)ext or (b)inary data?" This should be done using the technique described in the section in Project 2 entitled Reading Input from the Keyboard.

If the user answers "t" to this question, MDb will ask the user for a series of five text files to load.

MDb will first ask the user for a file in which movie data is stored. This should be done using the technique described in the section in Project 2 entitled Reading Input from the Keyboard. MDb will then read in the specified data file and store the data. Each line in this data file describes a particular movie. The format of this file is the same as for Project 1. Note that while there may be more than one movie in this database with the same title, the title together with the information in the first set of parentheses after the title (release year and possible Roman numeral, as described in Project 1) will be unique.

MDb will then ask the user for a file in which series data is stored. This should also be done by reading input from the keyboard. MDb will then read the specified file and store the data. The series data file format is described in Project 2 in the section entitled Input Format.

After the series data file, MDb will then ask the user for the following three data files in turn – actor, director, and producer. This should also be done by reading input from the keyboard. After each of these files is specified, MDb will read the specified file and store the data in a **LinkedHashMap** that is keyed on the media makers' names. In addition, any new movies, series, or episodes found in the actor, director, and/or producer files should be added to the existing lists of movies, series, etc. The actor, director, and producer data file formats are described below in the section entitled Input Format.

If, instead of choosing to load the data from text files, the user answers "b" to read the data from a binary file, MDb will ask the user for the name of a single binary file to read. Asking the user for this filename should also be done by reading input from the keyboard. After this file is specified, MDb will read the file and store the data. Because the format of this data file will be determined by the serialization methods, its format is not described in this document.

If any of the specified files are not found, MDb will ask for the names of those files again until such files are found. If the user enters return without entering a file name twice in a row, MDb will exit.

Once the data is loaded, MDb will enter a loop where it asks the user questions. As with the file name(s), answers to these questions should come from the keyboard as described in the section in Project 2 entitled Reading Input from the Keyboard. Depending on each answer given, MDb will proceed through a series of questions before displaying data to the user and starting over with the first question again.

The first question all users will be asked is "Search (m)edia or (p)eople?" If the user answers "m" to the first question, MDb will behave just as described in Project 2 (except with respect to the possibility of saving the data in either text or binary format, as described at the end of this section). If the user answers "p" to the first question, MDb will behave as described immediately below.

MDb will ask "Search for (e)xact or (p)artial matches?" Whether the user answers "e" or "p" to the second question, MDb will prompt the user for the name on which to search. After the name is entered, if the user chose to search for an exact match, MDb will use the name the user entered as the key on which to search the **LinkedHashMap** of media makers, whereas if the user chose to search for a partial match, MDb will search linearly through links of the **LinkedHashMap** for any name that contains the search string entered by the user.

If MDb finds no match for the name entered, it will inform the user of that fact and drop to the end of the main loop (to the "Continue (y/n)?" question discribed in Project 2). If MDb finds multiple matches to a partial name search, it will display the information on the matching media makers in the format described below in the section entitled "Output Format." If MDb find one media maker in its search (either exact or partial), it will ask "Display (t)ext or (g)raph?" If the user enters "t" to this question, MDb will display the information on the matching media maker in the format described below in the section entitled "Output Format" (as a special case of the display described for multiple matches). If the user enters "g" to this question, MDb will ask "Display (p)ie chart or (h)istogram?" Depending on the answer the user gives to this question, MDb will display one of the graphical representations of the data described below in the section entitled "Graphical Display."

After the data is displayed to the users (either in text or graphically), MDb will drop to the "Save (y/n)?" question described in Project 2. However, if the user enters "y" to this question, MDb will ask "Write (t)ext or (b)inary data?" If the user chooses "t," MDb will prompt for a file name to which the output should be saved, then save the data in a file with the given name in the same format as it was displayed to the user. If the user chooses "b," MDb will also prompt for a file name to which the output should be saved, then save the data in a file with the given name using object serialization.

Graphical Display:

Producing graphical displays of information can be very useful to users. Therefore, your program will have the ability to display pie charts and histograms to the user to display the data. When the user selects option "g" for graph, as described above, a graphical display of the chosen type will be displayed to the user, based on the data for the media maker found in their name search.

If the user has chosen "p" for pie chart, MDb will display a pie chart with up to six slices, one slice for each of movie acting credits, series acting credits, movie directing credits, series directing credits, movie producing credits, and series producing credits, with the width (subtended angle) of each slice proportional to the percent of credits for that movie maker for that category. Each slice should be labeled with the credit type that corresponds to that region. If the movie maker has no credits in some category, the slice for that category should be omitted. For example, if someone has only movie acting credits and no others, then the entire chart will be one big 360° slice (a circle) labeled "Movie Acting."

If the user has chosen "h" for histogram, MDb will display a histogram of the movie maker's credits over time, with one bar per year. The first year in the histogram will be the first year for which the person has a credit. The last year in the histogram will be the last year for which the person has a credit. The height of each bar will be proportional to the number of credits that person has in that year. The bar itself will be subdivided into horizontal sections representing the six categories of credits listed above for pie charts. The height of each subdivision should likewise be proportional to the number of credits that person has in that category in that year. If a person has no credits in a given year for a given category, that category should be omitted from the given bar. If a person has no credits at all in a given year, that bar should be omitted from the histogram. However, the year itself should still be represented.

Each graphical display should have a title at the top clearly indicating the data it represents (e.g., "Angelina Jolie – Percentage of Credits" or "Angelina Jolie – Credits over Time." Other details of the graphical displays (such as colors used) are up to you.

Learning Objectives:

Hash Maps:

You have already dealt with lists as a basic data structure for storing and retrieving data. Hash maps are an alternate way to quickly store and retrieve data. Java provides the **LinkedHashMap** class (among others) which has this functionality. In this project you will use a **LinkedHashMap** for saving and retrieving information on media makers. The keys in this map will be the names of the media makers. The values will be the media maker objects themselves. Note that we are using a **LinkedHashMap** so that you can also search linearly through the data structure to match using partial names.

Input/Output Formats:

Text Input Formats

Movie and Series Text Input Formats

The text input file format for movies is the same as the input file format from Project 1, with the exception that movies can also indicate that they have been suspended by the appearance of the word "SUSPENDED" in two sets of curly braces after the release year of the movie. See that project for additional details. The text input file format for series is the same as the input file format specified in Project 2, with the exception that an episode number may alternately be given as YYYY-MM-DD (that is, four digits indicating the year, a dash, two digits indicating the month, a dash, and two digits indicating the day). See that project for additional details.

Actor Text Input Format

The actor text file will consist of a series of lines for one actor, followed by a blank line, followed by a series of lines for another actor.

The first line for a given actor will consist of the actor's "last name(s)" aka "family name(s)," followed by a comma and a space, followed by the actor's "first name(s)" aka "personal name(s)," possibly followed a disambiguation number (a Roman numeral in parentheses), followed by one or more tabs, followed by one of the actor's acting credits. Each subsequent line for the same actor will consist of one or more tabs followed by one of the actor's acting credits.

Each acting credit for a movie consists of the name of the movie, followed by a space, followed by the year that the movie was released (along with a possible slash and Roman numeral, if multiple movies with that name were released that year), followed by a space, optionally followed by a media type – either "(TV)" for made-for-TV movies or "(V)" for straight-to-video movies – optionally followed by a space and the word "SUSPENDED" in two sets of curly braces, followed by two spaces, optionally followed by one or more parenthetical notes about this credit, followed by the role the actor played in that movie contained in square brackets, optionally followed by the billing order of the actor in that movie in angle brackets.

Each acting credit for a series consists of the name of the series in double quotes, followed by a space, followed by the year that the series first aired in parentheses, followed by a space, followed by the episode name and/or parenthetical episode number in curly braces, optionally followed by a space and the word "SUSPENDED" in two sets of curly braces, followed by two spaces, optionally followed by one or more parenthetical notes about this credit, followed by the role the actor played in that episode of that series contained in square brackets, optionally followed by the billing order of the actor in that episode in angle brackets.

For example:

```
Watson, Emma (II)   'Harry Potter': Behind the Magic (2001) (TV)  [Herself]
      'Harry Potter': Behind the Magic (2005) (TV)  [Herself]
      …
      Conjuring a Scene (2004) (V)  (archive footage) (uncredited)  [Hermione Granger]
      …
      Harry Potter and the Deathly Hallows: Part 1 (2010)  [Hermione Granger]  <2>
      …
      "Entertainment Tonight" (1981) {(2014-03-05)}  [Herself]
      "Entertainment Tonight" (1981) {(2015-04-15)}  (archive footage)  [Herself]
      …
```

The first line shows that the actor's name is "Emma Watson" and she is distinguished from any other Emma Watson who may be represented in the data by the Roman numeral "II" that follows her name. That line also shows that she appeared in a made-for-TV movie called "'Harry Potter': Behind the Magic" that was released in 2001 and that she played herself in that movie.

The second line shows that she also appeared in an identically named made-for-TV movie in 2005 and also played herself.

The third line given shows that Emma Watson also appeared in a straight-to-video movie called "Conjuring a Scene" that was released in 2004. In that movie she played the role of "Hermione Granger" although the parenthetical notes indicate that this was through the use of archive footage and that she did not appear in the movie's credits.

The fourth line given shows that Emma Watson appeared in "Harry Potter and the Deathly Hallows: Part

1" which was released in 2010. In that movie, she played the role of "Hermione Granger" and was the second billed actor in that movie.

The fifth line given shows that she appeared in the March 5, 2014 episode of the "Entertainment Tonight" series that began broadcasting in 1981. In that episode, she appeared as herself.

Finally, the six line given shows that she also appeared in the April 4, 2015 episode of "Entertainment Tonight" where she again appeared as herself but this time in archive footage.

Director and Producer Text Input Format

The format for the director and producer files is similar to the format for the actor file but somewhat simpler. In particular, there are no listings for role played in square brackets and no listings on billing order in angle brackets.

Text Output Formats

The output format for searches for movies and/or series (with or without episodes), will be as is given in Project 2. The text output for movie makers will be two lines describing the search performed – the first one indicating that the search was for people, the second line indicating whether the search was exact or partial and the name searched for – and a third line separating the search parameters from the results, similar to what was required for Project 2.

This would be followed by groups of lines listing the data for each person found, starting with one line for the full name of each matching person, followed by one line listing the keyword "ACTING," followed by one line for each acting credit (in alphabetical order) in the same format as for the movies/series/episodes output, followed by one line listing the keyword "DIRECTING," followed by one line for each directing credit (in alphabetical order) in the same format as for the movies/series/episodes output, followed by one line listing the keyword "PRODUCING," followed by one line for each producing credit (in alphabetical order) in the same format as for the movies/series/episodes output. This data should then be separated from the group of lines for the next matching person (if any) by a line of eighty minus signs.

For example:

```
SEARCHED PEOPLE
PARTIAL NAME: Angelina
================================================================================
Angelina Abdullaeva
MOVIE: This Is Andromeda (2015)
--------------------------------------------------------------------------------
…
Angelina Jolie
ACTING
MOVIE (TV): 10 Years of Tomb Raider: A GameTap Retrospective (2008)
…
MOVIE (TV): WWE Tribute to the Troops (2014)
DIRECTING
MOVIE: A Place in Time (2007)
…
MOVIE: Unbroken (2014/I)
PRODUCING
MOVIE: Africa (2015)
…
MOVIE: Unbroken (2014/I)
```

How to Complete this Project:

Preliminary Design:

1 During the lab session and in the week following, you should work with your partner(s) to determine the classes, variables, and methods needed for this project and their relationship to one another. This will be your preliminary design for your software.

1.1 Be sure to look for nouns in the project description. More important nouns describing the items of interest to the "customer" should probably be incorporated into your project as classes and objects of those classes. Less important nouns should probably be incorporated as variables of the classes/objects just described.

1.2 Be sure to look for verbs in the project description. Verbs describing behaviors of the desired objects and the systems as a whole should probably be incorporated into your project as methods.

1.3 Also look for adjectives, if any, in the project description. Adjectives often describe features of objects that could be incorporated into your project as interfaces to be instantiated by your classes.

1.4 Write down these nouns, verbs, and adjectives (if any), along with their corresponding classes/variables, methods, and interfaces (if any).

1.5 Next, use UML class diagrams as tools to help you establish proper relationships between your classes, variables, methods, and interfaces (if any).

2 Once you have completed your UML design, create Java "stub code" for the classes specified in your design. Stub code is the translation of UML class diagrams into code. It will contain code elements for class names (potentially including names for abstract classes and interfaces), variable names, and method names; relationships between classes such as inheritance, composition, and aggregation as appropriate; variable types; and method signatures. Stub code does not, however, contain method bodies (except for return statements for methods that return values or object references – these should return placeholders such as `null`). Because we are using object-oriented design, in which the inner workings of one class are largely irrelevant to the classes with which it interfaces (that is, we are using encapsulation), we do not need to complete the implementation of any class until the design is complete.

3 Add comments to your stubbed code as specified in the documentation requirements posted on the class website. Run your commented stubbed code through Javadoc as described in the Lab 2 slides. This will create a set of HTML files in a directory named "docs" under your project directory.

4 Create unit tests using JUnit for all of the non-trivial units (methods) specified in your design. There should be at least one test per non-trivial unit and units with many possible categories of input and output should test each category. (For example, if you have a method that takes an argument of type `int` and behaves differently based on the value of that `int`, you might consider testing it with a large positive `int`, and small positive `int`, zero, a small negative `int`, and a large negative `int` as these are all likely to test different aspects of the method.)

5 At the end of the first week, you will turn in your preliminary design documents (see ***Due Dates and Notes***, below), which the TA will grade and return to you with helpful feedback on your preliminary design. **Please note:** You are encouraged to work with the instructor and the TAs during their office hours during the design week to get feedback throughout the design process as needed.

Final Design and Completed Project

6 Using feedback from the instructor and TAs as well as your own (continually improving) understanding of OO design, revise your preliminary UML design.

7 Make corresponding changes to your code, including its comments.

8 Make corresponding changes to your unit tests.

9 Create a new set of Javadoc files using Eclipse and inspect them to make sure your final design is properly documented in your source code.

10 Implement the design you have developed by coding each method you have defined. A good approach to the implementation of your project is to follow the project's milestones in the order they have been supplied. If you find that your design does not allow for the implementation of all methods, repeat steps 6, 7, 8, and 9.

11 Test each unit as it is implemented and fix any bugs.

12 Test the overall program and fix any bugs.

13 Submit your project (see ***Due Dates and Notes,*** below).

Extra Credit Features:

You may extend this project with more search features for an extra 5 points of credit. Think of ways to enable a wider range of searches to be used, such as searching based on regular expressions or wild cards. Alternatively, think of ways to decompose one of the classes into logical subclasses. You could also revise user interface elements. If you revise the user interface, you **must** still read the file name from the keyboard and the data from the text files.

To receive the full five points of extra credit, your extended feature must be novel (unique) and it must involve effort in the design and integration of the feature into the project and the actual coding of the feature. Also, you must indicate, on your final UML design, the portions of the design that support the extra feature(s); and you must include a write-up of the feature(s) in your milestones file. The write-up must indicate what the feature is, how it works, how it is unique, and the write-up must cite any outside resources used. If you create any non-trivial units in your extra credit work, you must create appropriate unit tests for them.

Due Dates and Notes:

Note that both the preliminary design and the final project are to be submitted electronically. Paper copies will not be submitted. The UML should preferably be in PDF format, although high resolution PNG or JPG would be acceptable alternatives. The list of nouns, verbs, and adjectives and their corresponding classes/variables, methods, and interfaces should be in PDF format.

Your preliminary design (list of nouns, verbs, and adjectives; UML; stub code; detailed Javadoc documentation; and unit tests) is due on **Monday, 7 March 2016**. Submit the project archive following the steps given in the submission instructions **by 11:00pm**.

The final version of your project including final design (UML, Javadoc, unit tests) and final implementation and milestones file is due on **Monday, 28 March 2016**. Submit the project archive following the steps given in the submission instructions **by 11:00pm**.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do

not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

As noted in the syllabus, you are required to work on this programming assignment in a group. The group should turn in only one (1) copy of the assignment. This should contain the names and student ID numbers of **all** group members on the cover sheet.

Each group member is required to contribute equally to each project, as far as is possible. You must thoroughly document which group members were involved in each part of the project. For example, if you have three methods in your program and one method was written by group member one, the second was written by group member two, and the third was written jointly and equally by group members three and four, your cover sheet must clearly indicate this division of labor. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.

When zipping your project (or design) for submission, be sure to follow the instructions carefully. In particular, before zipping the project be sure to:

- **Place additional files (such as UML diagrams, cover sheets, and milestones files) within the "docs" directory inside your Eclipse folder for the given project and be sure that Eclipse sees these files (look in the Package Explorer and hit Refresh if necessary).**
- **Use the Eclipse export function to compress all files into .zip format. Other formats will not be accepted.**
- **Use one of the following formats for the UML: .png .jpg or .pdf. Custom formats such as .uml or .dia are NOT acceptable. If you are unsure how to export the file in that format, take a screen-shot of the diagram and include that.**