Project 4

Computer Science 2334
Spring 2016

This project is group work. Group composition will be determined later. 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 User Interface****."*

Milestones:

1. Create appropriate classes, complete with data fields and methods, to handle application data on movies, series, and their makers as described below under Model. *15 points*
2. Add a class, complete with data fields and methods, to the model to allow the model to correctly interact with the controller and the views. *10 points*
3. Create appropriate classes, complete with data fields and methods, for the views described below. *25 points*
4. Create an appropriate class, complete with data fields and methods, for the controller as described below. *25 points*

- ► Develop and use a proper design. *15 points*
- ► Use proper documentation and formatting. *10 points*

Description:

An important skill in software design is extending the work you have done previously. For this project you will rework Project 3 in order to handle information pertaining to movies, series, and their makers and allow users to view and manipulate that information graphically. This project will be organized around the model, view, controller (MVC) design pattern which gives us a way to organize code involving graphical user interfaces (GUIs). In particular, you will create a single model to hold the data, several views to display and manipulate different aspects of the data, and one controller to moderate between user gestures and the model and views. For this program you may reuse some of the classes that you developed for your previous projects, although you are not required to do so. Note that much of the code you write for this program could be reused in more complex applications.

Model:

You will create a model class called "**MediaModel**." Models in the version of the MVC design pattern we have used in class contain data and methods for the application objects being modeled (which are all related to movies, series, and media makers in this project) as well as data and methods to allow the model to interact with views. Your model class will follow this version of the MVC design pattern.

For the application objects, you will have classes to represent the following kinds of things: movies, series, episodes, actors, directors, and producers. Note that you have represented these application objects in previous projects, although you will need to modify your classes for those objects at least slightly in order to satisfy the requirements of this project. (You may, of course, modify your classes substantially for this project, if doing so would substantially improve your project.)

As you are designing or revising the classes for these application objects, please note that it will be useful in this project (and in Project 5) to go from objects of one type to another and back again using references without having to search through lists or hashmaps. So, for example, you should consider not only having a list of movies within each actor but also a reference from each movie back to its actors.

The data for this model will enter the system through three alternate ways: (1) It may be read in from a text file, which we will refer to as *importing* the data. (2) It may be read in using object I/O, which will will refer to as *loading* the data. (3) It may be entered by a person using the input views described below, which we will refer as *entering* the data.

Whether data is imported or entered, it must meet the following reality checks: (1) An episode year may not be outside the range of years for the series of which it is a part. (2) There may not be two episodes with the same season and episode number pair. If invalid data is found during import or entry, the user will be informed of the error and the model will not be updated to include that data.

To interact with views, the **MediaModel** class will have variables and methods akin to those from the examples we have seen in MVC lectures and labs. In particular, when new information is added to the model by importing, loading, or entering new data, any relevant display views should be notified so that they may update themselves to reflect the new data.

Views:

Producing views of information can be very useful to users. Therefore your program will create and maintain several views of the data, as described below.

Selection View:

The *Selection View* will be displayed as soon as the program is started. There will be a title in the top bar that says "MDb." In addition, there will be a menu bar with a file menu, an edit menu, a display menu, and, in the content pane of the window, a vertical list of radio buttons and a vertical scrolling list of data displayed side by side.

Content Pane:

Within the content pane of the selection view, there will be two main panels. The left panel will contain a vertical set of radio buttons labelled "Media," "Movies," "Series," "Episodes," "Makers," "Actors," "Directors," and "Producers." The right panel will contain a vertical list of data, reflecting all the data in MDb of the type selected by the radio button on the left. (For example, if the radio button for movies is selected in the left panel, only movies will be displayed in the list in the right panel. However, if the radio button for media is selected in the left panel, all media in MDb will be displayed in the right panel.) Note that there will be no data in MDb until some is imported, loaded, or entered, so initially the list in the right panel will be empty. At the top of the left panel will be the title "Selection." At the top of the right panel will be a title for the type of data displayed within the list (e.g., "Media" or "Movies"). The list will be kept in sync with the underlying model information on movies, series, etc. Moreover the list will be kept ordered alphabetically.

File Menu:

The file menu will be marked "File" and have entries for "Load," "Save," "Import," and "Export." Initially, only Load and Import will be active. Save and Export will be grayed out and inactive until at least one object (movie, actor, etc.) has been added through the GUI or an existing one has been read in using Load or Import. In general, menu items should only be active when executing the action would be appropriate. (For example, it makes no sense for the user to save or export data if none is present.)

If the user chooses Load or Import, MDb will present the user with a file picker and allow selection of one or more files to read in via object input or text input, as appropriate. Additionally, if the user chooses Import, MDb will ask which type of data the file contains (movie, series/episode, actor, director, or producer).

Once data is present in the system, the Save and Export menu items will become active. If the user chooses Save or Export, MDb will present the user with a file picker and allow designation of one or more files for writing via object output or text output, as appropriate. If the user chooses Export, MDb will ask which type of data to export (movie, series/episode, actor, director, or producer).

For each inactive menu item, if the user hovers the pointer over that component, a tool tip will appear to let the user know why the button is not active. (For example, "Cannot export data; no data present.")

Edit Menu and Associated Views:

The edit menu will be marked "Edit" and have entries for "Add," "Edit," "Delete," "Clear," and "Clear All." As with the file menu, menu items should be grayed out and inactive until they are appropriate. Since only "Add" is appropriate until data is entered, it will be the only one initially active. Moreover, it will only be active if one of the following radio buttons is selected: "Movies," "Series," "People," "Actors," "Directors," or "Producers."

If the user selects Add, a *data entry view* will appear and provide the user an interface to allow data entry of the same type as selected by the radio button. For example, if the Movies radio button is selected, a *Movie Entry View* (see below) will appear. Once the user fills in the data to the view, the controller will take the data entered and pass it along to the model, asking the model to add an object of the appropriate type. However, note that the model will not allow duplicate names for people or duplicate names and years for movies to be added to the database and the user should be notified by the view if any such data entry error is made and allowed to add a disambiguation number or to discard the data.

If the user selects an entry from the main scrolling list (in the content pane of the Selection View) and clicks Edit, then an *edit view* will appear to allow the user to edit the existing data on the selected object. This view will be similar to the add dialog for that object type, except that the fields will be pre-populated with the data currently found in the model.

If the user selects more than one object from the main scrolling list and clicks Edit, then a series of edit views will appear, one for each selected object.

If the user selects one or more objects one of the main scrolling lists and clicks Delete, then a dialog will appear to confirm whether the user wants to delete the selected object(s) and its (their) data. Note that if an object is deleted from the system, all references to that object should be deleted from the system. So, for example, if a movie is selected for deletion from the system, not only will the associated movie object be deleted but all acting, directing, and producing credits referring to that movie will be altered to remove that movie from them.

If the user selects Clear, then a dialog will appear to confirm whether the user wants to clear (delete) all of the data of the type appearing in the main scrolling list. As with delete, if an object is deleted from the system, all references to that object should be deleted from the system. So, for example, if movies are selected for clearing from the system, not only will the associated movie objects be deleted but all acting, directing, and producing credits referring to all movies will be deleted from the system.

Similarly, if the user selects Clear All, then a dialog will appear to confirm whether the user wants to clear (delete) all data of all types from the system.

There will be add and edit views for the following object types: Movies, series, episodes, people, actors, directors, and producers. Each view should allow the user to enter all of the data fields found in the text files and described in previous projects for the corresponding type of object. For episodes, the user should be presented with a list of series to chose from and the user must select exactly one series to which the episode will belong. For actors, directors, and producers, MDb will present the user with lists of movies and episodes to choose from to add to their acting, directing, and producing credits, respectively. Here, however, the user may choose any number up to all of the movies and/or episodes present in the system, including zero. For people, the user should be able to add acting, directing, and/or producing credits. For all of these credits, it should also be possible for the user to choose to create a new movie or episode "on the fly" while entering credits. (For example, the Actor Entry View should give the user the option to open a Movie Entry View and/or an Episode Entry View to enter a new movie or episode while in the middle of adding an actor to the system.)

Display Menu and Corresponding Display Views:

The display menu on the Selection View will be marked "Display" and have entries for "Pie Chart," and "Histogram."

These views will be very similar to the graphical output seen in Project 3 with three differences: (1) The data displayed will be based on the current selection(s) in the content pane. For example, if the user has highlighted two people in the content pane and selects Pie Chart, then the user will be presented with two pie charts (one for each person). (2) The data displayed will be based on the type of data shown in the content pane. For example, if the user has selected the radio button for Actors, so that only actors are shown in the scrolling list in the content pane, and selects Pie Chart, then any pie charts created in response to this request will be limited to acting credits – they will not contain directing or producing credits. (3) If the data in the model is updated, the graphical output should change to match the new model data.

Notes on Views:

Note that there will be one Selection View for your program. It will be opened when your program runs. When the user closes this view, your program should exit. In contrast, there may be many Display Views open at any given time. Each time the user selects a display menu item, one or more new Display Views should open. Each Display View can be closed independently by the user by closing the window in which it resides. Additionally, if all of the data with which a particular Display View is associated are deleted, the view should automatically close itself.

Controller

Besides at least one model and at least one view, every GUI-based program using the MVC design pattern needs to have at least one controller. The controller is responsible for connecting the model(s) to the view(s) so that appropriate actions are taken in response to user gestures and that appropriate representations of data are presented to users.

For this project, the appropriate actions for user gestures and appropriate representations of data are described above, under the individual views. The controller that you create, then, will need to ensure that the actions are taken and views updated as described above. Call this controller **MediaController**.

Text Input and Output

The format of the text files containing data will be the same as for Project 3. **These formats should be used for both importing and exporting data.**

How to Complete this Project:

1 During the lab session and in the week following, you should create figures, on engineering paper or using drawing software, to show approximately what each view will contain. These figures do not need to exactly match the appearance of the final windows but should contain all major components and show their basic layout. This will be a part of your preliminary design for your software.

2 During the lab session and in the week following, you should determine the classes, variables, and methods needed for this project and their relationships to one another. This will be the other part of your preliminary design for your software.

2.1 Make a list of the nouns you find in the project description that relate to items of interest to the "customer." Mark these nouns as either more important or less important. 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. This list will be turned in with your preliminary and final designs long with your other design documents.

2.2 Make a list of the verbs you find in the project description that relate to items of interest to the "customer." Verbs describing behaviors of the desired objects and the systems as a whole should probably be incorporated into your project as methods. This list will be turned in with your preliminary and final designs long with your other design documents.

2.3 Make a list of 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. This list will be turned in with your preliminary and final designs long with your other design documents.

2.4 Relate the nouns, verbs, and adjectives (if any), to classes/variables, methods, and interfaces (if any) and list them. This list will be turned in with your preliminary and final designs long with your other design documents.

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

3 Once you have completed your UML design, create Java "stub code" for the classes and methods specified in your design. Stub code is the translation of UML class diagrams into code. It will contain code elements for class, variable, 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 any class until the design is complete.

4 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.

5 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.)

6 At the end of the first week (see ***Due Dates and Notes***, below), you will turn in your preliminary design documents **including your view figures**, 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

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

8 Make corresponding changes to your stub code, including its comments.

9 Make corresponding changes to your unit tests.

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 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 Once you have completed the project and are ready to submit it for grading, 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.

14 Submit all parts of your completed project. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

Extra Credit Features:

You may extend this project with more features for an extra 5 points of credit. For example, you could think of ways to present the user with helpful information about the program, such as a context-dependent help system.

To receive the full five points of extra credit, your extended features must be novel (unique) and must involve effort in the design of the extra features and their integration into the project and the actual coding of the features. Also, you must indicate on your final UML design which portions of the design support the extra feature(s); and you must include a write-up of the feature(s) in your milestones.txt file. The write-up must indicate what each feature is, how it works, how it is unique, and the write-up must cite any outside resources used.

Due Dates and Notes:

Your preliminary design (**view figures**; list of nouns, verbs, and adjectives; UML; stub code; detailed Javadoc documentation; and unit tests) is due on **Friday, 1 April 2016**. Submit the project archive following the steps given in the submission instructions **by 11:00pm**. **Submit your view figures by photographing or scanning your hand drawn figures or exporting to png or pdf figures made using drawing software, then place them in the doc directory before creating your project archive for submission.**

The final version of your project including final design (UML, Javadoc, unit tests) and final implementation is due on **Monday, 18 April 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 team. The team should turn in only one (1) hard copy and one (1) electronic copy of the assignment. Both the electronic and hard copies should contain the names and student ID numbers of **all** team members on the cover sheet. If your group composition changes during the course of working on this assignment (for example, a group of five splits into a group of two and a separate group of three), this must be clearly indicated in your cover sheet, including the names and student ID numbers of everyone involved and details of when the change occurred and who accomplished what before and after the change.

Each team member is required to contribute equally to each project, as far as is possible. You must thoroughly document which team members were involved in each part of the project. For example, if you have three functions in your program and one function was written by group member one, the second was written by team member two, and the third was written jointly and equally by team members three and four, your cover sheet must clearly indicate this division of labor. Giving improper credit to team 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 view figures, 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),**
- **compress all files into a .zip format. The formats .rar and .7z will no longer be accepted. Also, when submitting the initial design make sure that the UML is in one of the following formats: .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 attach that, and**
- **rename the project folder to the 4x4 of the team member submitting the project. Note that renaming the project folder to your 4x4 *before* zipping is not the same thing as naming the zip file with your 4x4. The latter is fine; the former is *mandatory*.**