Project 5

Computer Science 2334
Spring 2015

This project is individual work. Each student must complete this assignment independently.

User Request:

"Expand TeamMate by adding "separation search" and exception handling."

Milestones:

1	Create appropriate classes, variables, and/or methods to deal with the concept of degrees of separation, as described below.	*15 points*
2	Create appropriate classes, variables, and/or methods to recursively search a player's teammates for their teammates, as described below.	*25 points*
3	Create appropriate classes, complete with variables and methods, for the new views described below.	*5 points*
4	Revise as necessary the classes, including variables and methods, of the existing model, controller, and views to allow them to correctly interact with the new methods and views.	*15 points*
5	Improve the robustness of your code by adding exception handling as described below.	*15 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 build on Project 4 in order to add new functionality related to the concept of degrees of separation between players and improving the exception handling for I/O.

Degrees of Separation:

For this project, we are going to define degree of separation *DS* betwene players informally, as follows.

DS (*A*, *A*) = 0

DS (*A*, *B*) = 1 if *A* and *B* played for the same team in the same year.

DS (*A*, *C*) = 2 if *A* and *C* did not play for the same team in the same year but there is some player *B* for which *DS* (*A*, *B*) = 1 and *DS* (*B*, *C*) = 1.

Etc.

Even more informally, degree of separation simply means the number of steps it takes to get from one player to another, counting other players as steps if they played on the same team in the same year.

Models, Views, and Controllers:

As with Project 4, 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 one or more models to hold the application data and extend them to act as sources for communicating with views, multiple views to display and manipulate different aspects of the data, and one or more controllers to moderate between user gestures and the model(s) and views.

The models, views, and controllers described in Project 4 all need to be present and functioning in Project 5. See Project 4 for a description of those elements. For this program you could reuse some or all of the classes that you developed for your Project 4, although you are not required to do so. If your Project 4 was designed and implemented well, you should be able to use those classes with little or no modification. If your project 4 had significant design or implementation problems, you will need to significantly change that code as well as added new code for the new functionality. Note that if you did poorly on Project 4, this gives you a chance to redo your Project 4 and potentially gain credit for points you missed previously.

In addition to the Project 4 features, Project 5 will add new features related to degrees of separation. In particular, one new view will be added and the Selection View will be modified slightly.

The new view is described below. Like the pie chart and mapping views of Project 4, this new view should persist until closed by the user or until the data it displays has been cleared from the system. While it is open, it will not interfere with accessing other views. If the user modifies data in the model(s) while this view is open, this view should update itself if the data relevant to it has been changed.

Degrees of Separation List View:

The *Degrees of Separation List View* will be a simple list of all of the steps from one player to another. This view will be tied to a new button in the *Selection View*. The new button will be labeled "Separation" and will be below or alongside the Add, Edit, and Delete buttons for players.

When there is one player highlighted on the player list and this button is clicked, one or more *Degrees of Separation List Views* will appear, each showing the name of the player followed by a year and a team name. A separate *Degrees of Separation List View* should appear for each season in the data for the selected player.

When two players are highlighted on the player list and this button is clicked, one or more *Degrees of Separation List Views* will appear, each showing a list. At the top of each list will be one of the highlighted players. At the bottom of the list will be the second highlighted player. Between them will be the minimum number of other players necessary to find a team connection between the two. Beside each player, except for the last, will be the year and team name for their connection to the next player on the list. There will be one such Degrees of Separation List View for each such minimal connection. For example, if players A and B are highlighted and they played together on the same team in five different years and on a different team in two additional years, there should be a total of seven views, each showing player A's name at the top with the shared year and team name next to it, and B's name at the bottom. As a second example, if players A and Z are highlighted but A and Z never played for the same team in the same year and the shortest connection that TeamMates can find for A and Z goes through players B, C, D, … Y, then a view should appear that shows A at the top of the list alongside the year and team name shared with B, then B should appear second on the list alongside the year and team name shared with C, and so on until Z appears at the bottom of the 26 line list. If additional 26 step connections are also found between A and Z, those lists should appear in separate *Degrees of Separation List Views.*

In building up the lists for these views, TeamMates should search recursively through the data.

Exception Handling:

Because I/O errors are likely to occur without warning, all of your I/O routines should be thoroughly wrapped with exception handling routines. Note that this does not mean simply re-throwing all possible I/O exceptions. Instead, you should consider the possible I/O exceptions individually and write

appropriate exception handling routines to provide for graceful handing of different exception types. For example, a `FileNotFoundException` could be handled by asking the user to select an alternate file or select not to do I/O at this time.

How to Complete this Project:

1 During the lab session and in the week following, you should work ~~with your partner(s)~~ to 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 deisgn for your software.

2 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.

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 of 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 view figures and preliminary 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. If you find that your design does not allow for the implementation of all methods, repeat steps 5 and 6.

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, April 24th**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**. 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 **Friday, April 31st**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**.

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.

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 page, 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), and**
- **rename the project folder to your 4x4. 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*.**