Project 4

Computer Science 2334
Spring 2015
This project is group work. Each group must have at least two members.

User Request:

"Create a sortable and searchable data system for places, people, and teams ***with a Graphical User Interface****."*

Milestones:

1.	Create appropriate classes, complete with data fields and methods, to handle application data on places, people, and teams 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 places, people, and teams 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 "**TeamMateModel**." 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 places, people, and teams 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: players, cities, states*, and teams. 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.)

*Note that for the purposes of this project, we will consider the District of Columbia (DC) to be a state.

One difference is that team data now includes multiple years. For each year, most aspects of a team's data may change, including name, location, and roster. For this reason, teams will now be identified primarily by a team ID, which stays constant, even if other aspects of the team changes. Correspondingly, personal data now includes data for multiple years. In particular, while a person's name, birthplace, and birth date remain constant, each year each person may leave one team and join another. On the other hand, city and state data are taken as constant.

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 cities within each state but also a reference from each city back to its state. Similarly, you should consider not only having a list of players within each team but also a reference from each player back to its team.

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 sanity checks: (1) A person may not play for two or more different teams in the same year. (2) A person may not play for a team before he was born or after he has died. (3) A person may not have died before he was born. 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 **TeamMateModel** 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 "TeamMate." In addition, there will be a menu bar with a file menu, a graph menu, and, in the content pane of the window, three vertical lists displayed side by side each with its own title above it and set of buttons below it.

Content Pane:

Within the content pane of the selection view, the three vertical scrolling lists will be organized from left to right. The first list will be titled "Places," the second list will be titled "People," and the third list will be titled "Teams." Below each list will be a set of buttons labeled "Add, "Edit," and "Delete."

Initially, all lists will be empty and most buttons will be grayed out and inactive. The only initially active button will be the Add button below the places list. This is because, to add a birthplace for a person or a location for a team, the user will select from a menu of available places, which would be empty if no places exist in the system.

The lists will be kept in sync with the underlying model information on people, places, and teams and

the buttons will become active and inactive as appropriate. For example, once there is at least one place, the Edit and Delete buttons below the place list would be appropriate and should become active, but each Edit and Delete button below each other list should remain inactive until that list contains at least one entry. Similarly, some menu items in the file menu will initially be grayed out and inactive.

Add, Edit, and Delete Buttons and Associated Views:

If the user clicks the Add button below the place list, a *state selection view* will appear to ask the user to select a state from a scrolling list that shows the abbreviations of all 50 states plus the District of Columbia (DC). Once a state has been selected, a *city entry view* will appear to ask the user to enter the name, latitude, and longitude of the city using text fields. The controller will take the information entered for the city and pass it along to the model, asking the model to add the city to the selected state. However, note that the model will not allow duplicate city names to be added to the same state and the user should be notified by the view if any such data entry error is made. Once at least one city has been added to TeamMate, the Add button for people on the content pane will become active. (Note that the Add button for team will only become active when there is at least one city and one person in the system.)

If the user clicks the Add button below the people list, an *add person view* will appear to ask the user for the person's name, birth date, birthplace, and date of death (which should be marked as optional). The name should be entered through a text field. The birthplace should be chosen by having the user first select a state from scrolling lists of states abbreviations (using a state selection view), then select a city from a scrolling list of the cities within the selected state (using a *city selection view*). You may chose the method of entry for the birth date and (option) death date. The controller will take the information entered for the person and pass it along to the model, asking the model to add the person. However, note that the model will not allow duplicate personal names to be added and the user should be notified by the view if any such data entry error is made.

If the user clicks on the Add button below the team list, an *add team view* will appear to ask the user for the team's ID, which should be entered through a text field. The controller will take the ID entered for the team and pass it along to the model, asking the model to add the team. However, note that the model will not allow duplicate team IDs to be added and the user should be notified by the view if any such data entry error is made.

If the team is successfully added to the model, an *add season view* will appear, asking the user to enter a year, a city, and players. The year should be entered using a *year selection view*, which is a scrolling list starting with 1946 and running to the present year. The city and players should be selected from scrolling lists that only show available people (those not already assigned to any team for the given year).

As entries are added to each of the three main scrolling lists (for places, people, and teams), these will be kept ordered as follows: The places will be alphabetical by state abbreviations then by city name (that is, all of the Alabama cities will appear first in alphabetical order, then all of the Alaska cities, etc.), displayed as City Name, ST (where City Name is the name of the city and ST is the state abbreviation). The people will be alphabetical by last name then first name displayed as Last Name, First Name Middle Name(s). The teams will be alphabetical by team name.

If the user selects an entry from one of the main scrolling lists and clicks the Edit button below that list, 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. In addition, the team edit view will allow the user to click a button marked "Add Season" to bring up an add season view (as described above). Because the model will not

allow for duplicate seasons for a team, the year selection view should not include options for any years for which the team already has seasons.

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

If the user selects one or more objects one of the main scrolling lists and clicks the Delete button below that list, 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 person is selected for deletion from the system, not only will the associated person object be deleted but all team rosters referring to that person will be altered to remove that person from them.

File Menu:

The file menu will be marked "File" and have entries for "Load TeamMate Data," "Save TeamMate Data," "Import TeamMate Data," and "Export TeamMate Data." Initially, only Load TeamMate Data, and Import TeamMate Data will be active. Save TeamMate Data and Export TeamMate will be grayed out and inactive until at least one object (place, person, or team) has been added through the GUI or an existing one has been read in using Load TeamMate Data or Import TeamMate Data. As with the buttons, the 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 TeamMate data if none is present.)

If the user chooses Load TeamMate or Import TeamMate and there is unsaved data, your program will present a dialog box asking the user whether that data should be saved, exported, or discarded. The actions taken by your program in response to this save/export/discard dialog box should be to save, export, or discard this data, as specified by the user. Once there is no unsaved data in your system, your program 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. Once the data is read in, if it contains places, the places list will be populated with places from the model and the Edit and Delete buttons for places will become active, and similarly for the other types of objects. Note that, for text input files, TeamMate should ask for the name of the place file first, and only move on to ask for the name of the people file once there is at least one place in the system. It should approach teams similarly.

Once TeamMate data is present in the system, the Save TeamMate and Export TeamMate menu items will become active. If the user chooses Save TeamMate or Export TeamMate, your program 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.

For each inactive button or 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 currently active. (For example, "Cannot delete team(s); empty team list" or "Cannot delete team(s); no team(s) selected.")

Graph Menu and Corresponding Display Views:

The graph menu on the Selection View will be marked "Graph" and have entries for "Pie Chart" and "Map."

These views will be very similar to the graphical output seen in Project 3 with two 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 teams, three people, and four cities in the content pane and selects Map, then the user will be presented with two year selection views (one for each team) that allow for selection of multiple years. Once the user has selected year(s) for each team, the map will display each team's location for

each year, the birthplaces of the people who were associated with those teams in those years, the birthplaces of the three people selected, and the four cities selected. (2) 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 graph menu item, a new Display View 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 TeamMate 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 for this project, then, will need to ensure that the actions are taken and views updated as described above. Call this controller **TeamMateController**.

Text Input and Output

The format of the people and places text files containing data will be the same as for Project 3. However, the team file format will change as follows. For each team there will be one line that specifies the team ID code. This will be a three letter code, followed by a colon (:). This will be followed by zero or more lines, each containing a year, a team name, a team city, a team state (as a two letter abbreviation), and a team roster. Each of these data items will be separated from the preceding item by a semi-colon and a space. This includes each team member on the roster. 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 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 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 **Wednesday, April 1st**. 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 **Wednesday, April 15th**. 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.

As noted in the syllabus, you are required to work on this programming assignment in a group of at least two people. It is your responsibility to find other group members and work with them. The group 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** group 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 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 functions in your program and one function 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 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*.**