Project 3

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 people, places, and teams 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 teams.	*10 points*
5.	Use **LinkedHashMaps** to save to and retrieve information on teams.	*10 points*
►	Develop and use a proper design. (See 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, implementing object serialization for input and output, using the Java **LinkedHashMap** class, and adding a graphical display. To help you locate the modifications from Project 2 to Project 3, the changes between the instructions of the two projects are highlighted here in yellow.

People have certain life history data, such as birthdate and birthplace. Likewise, places such as cities and states have data associated with them. For example, states contain cities and they are the birthplaces of people, while cities are found in states and are also the birthplaces of people. Note that the relation of cities to states is different from the relation of states to cities, because each city is found in only one states whereas each state contains multiple cities. For this project, you will create a system that models these relationships between people, cities, and states.

Besides the types of data described above, this project will include data of another type – data on teams. A *team* (in this sense) is an organization located in a particular city in a particular state with a roster of team members. For each team, your system will keep track of its name, the city and state where it is located, and its roster.

As with Project 1, you will put together several techniques and concepts learned in CS 1323 and some new techniques to make an application. This application will allow users to search through data on people, places, and teams. We will call this application *TeamMate*. Note that much of the code you write for this program could be reused in more complex applications, as we will see in later assignments.

Your software will first ask the user for the file(s) where data is stored. This should be done using the technique described in the section below entitled Reading Input from the Keyboard. It will then read in the specified data file(s) and store the data into TeamMate. The data will either be stored as a trio of text files or a single binary data file. The binary data file will be written and read using object IO. The first text data file will be the same as for Project 1. Each line in the data file describes a particular person, including that person's birthplace. The second data file will contain one line for each team, formatted as follows: Team name, team city, team state (as a two letter abbreviation), 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. The third data file will contain one line for each city, formatted as follows: City name, state (as a two letter abbreviation), latitude, longitude. Each of these data items will be separated from the preceding item by a semi-colon and a space.

Once the data is loaded, your program 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 entitled Reading Input from the Keyboard. Depending on each answer given, TeamMate will proceed through a tree 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 "People, Place, or Team?" If the user enters "People," then TeamMate will ask "Sort or Search?" If the user enters "Sort," TeamMate will ask "First or Last?" If the user enters "First," then TeamMate will display to the user a list of information on all of the people in the database, sorted by first name then last name then middle name. Examples: "George Washington" would be listed before "John Adams" because the first name "George" comes before the first name "John" in standard lexicographic ordering and last names would be ignored. "Andrew Jackson" would be listed before "Andrew Johnson" because the first name is "Andrew" for both entries and the last name "Jackson" comes before the last name "Johnson." "George Herbert Walker Bush" would be listed before "George Walker Bush" because the first and last names match and "Herbert Walker" comes lexicographically before "Walker.") Similarly, if the user enters "Last" (to the question "First or Last?"), TeamMate will display to the user a list of information on all of the people in the database, sorted by last name then first name then middle name. (Example: "John Adams" would be listed before "George Washington" because "Adams" comes before "Washington.")

If the user enters "Search" rather than "Sort," TeamMate will ask "Exact or Partial?" If the user enters "Exact," then TeamMate will prompt for an exact name, then search the database for a name that matches the input exactly and display information on that person to the user. (Example: If the user input "John Adams" then a person in the database named "John Adams" would match but a person in the database named "John Quincy Adams" would not.) If the user enters "Partial," then TeamMate will prompt for a partial name, then search the database for names that include the input anywhere within the name, and display information on all of the matching people to the user. (Example: If the user input "John" then that would match "John Adams," "John Quincy Adams," "John Tyler," "Andrew Johnson," etc.)

If the user enters "Place" rather than "People," then TeamMate will ask "State or City?" If the user enters "State," TeamMate will prompt the user for a two letter state abbreviation, then search the database for a state matching the given input, and display to the user an alphabetical list of all cities within that state showing information on all of the people in the database born in each city. If the user enters "City," TeamMate will likewise prompt the user for a two letter state abbreviation, then a city name, then search the database for a city with a name that matches the input exactly and display information on all of the people in the database who were born in that city.

If the user enters "Team" rather than "People" or "Place," then TeamMate will prompt the user for the name of a team, then search the database for a team matching the given input, and display to the user an alphabetical list of all the team members on the matching team.

In all cases, data will be displayed to the console as described in the section entitled Output Format. Once data is displayed to the user, the user will be asked "Save or Skip?" If the user chooses "Save," TeamMate 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. After saving or skipping, TeamMate will ask the user "Graph, Continue, or Exit?" If the user enters "Graph," TeamMate will ask "Age or Location?" Whether the user chooses age or location, TeamMate will display a graphical representation of the data, as described in the section entitled Graphical Display then ask the user "Continue or Exit?" If the user enters "Continue," TeamMate will return to the first question ("People or Place?"). If the user enters "Exit," TeamMate will thank the user for using TeamMate and exit gracefully.

If the user response to any question is incorrect (that is, does not allow TeamMate to proceed to the next step), then People should present the user with a polite message stating that and asking the user to give new input. (Examples: If the user inputs "Peeple" to the original question "People or Place?" then TeamMate should saying something like "You did not enter 'People' or 'Place.' Please enter one or the other." If the user searches on a name that does not exist in the database, TeamMate should say something like "I could not find a person matching the name you entered. Please enter a different name.")

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 maps to the user to display the data. When the user selects option "Graph" for Graphical Display, as described above, a pie chart or map will be generated and presented to the user. The type of pie chart or map will be determined by the type of data chosen by the user in response to the questions.

If the user has chosen to graph age, TeamMate will display a pie chart with one slice for each age in the selected data, with the width (subtended angle) of each slice proportional to the percent of people with that age in the selected data. Each slice should be labeled with the age that corresponds to that region.

If the user has chosen to graph location, TeamMate will display a plate carrée map projection of the US with points plotted on it for all of the cities in the selected data. Each point should be labeled immediately above it with the name of the corresponding city. In addition, it the user had selected team data, the city that corresponds to the team's location should be connected to each city that corresponds to a team member's birth cities with a line.

Each graphical display should have a title at the top clearly indicating the data it represents (e.g., "Team Location and Birthplaces of Team Members" or "Ages of Team Members from MN"). Other details of the graphical displays are up to you (colors, etc.).

Learning Objectives:

Sorting and Searching:

Sorting data can be useful to users because the output may be organized in a way that makes it easier to use. It can also be useful to software developers because it can improve the efficiency of their software. Consider finding information about a person based on name. If the data structure holding the data is unsorted, you need to do a linear search through it to find an entry. However, if the data structure is sorted based on name, you can do a binary search instead. A binary search will, in most cases, take far fewer comparisons to find the desired entry than a linear search. You should ensure that your program uses a binary search when searching for an exact name. (Why only for exact names? Answer this question in your writeup on the project.)

Note that a collection in Java can have at most one *natural ordering*. You should determine an appropriate natural ordering for people (which must implement **Comparable**) and define the `compareTo()` method(s) to use that ordering. The other sort option will need to be implemented using the `compare()` method that comes from implementing **Comparator**.

Hash Maps:

You have already dealt with lists as a basic data structure for storing and retrieving data. Hash tables 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 teams. The keys in this map will be team names. The values will be the teams themselves. Note that we are using a **LinkedHashMap** so that it can retain an alphabetical ordering by team name. You should consider how to get the data into the **LinkedHashMap** in that order.

Input/Output Formats:

Text Input and Output Format

The input file format for people for Project 3 is the same as the input file format from Project 1. See that project for details.

The output format for the data on a single person matches that of the input format. If data on more than one person is displayed, headings will precede it. For example, if the user asked for data on people born in Michigan ("MI"), the output should look like this:

```
MI:
Benton Harbor:
Quacy Maria Barnes,26/09/1976,Benton Harbor,MI
Detroit:
Markita Aldridge,15/09/1973,Detroit,MI
Daedra Janel Charles,22/11/1968,Detroit,MI
Jermaine Jackson,07/06/1976,Detroit,MI
Flint:
Tawona Alhaleem,17/10/1974,Flint,MI
Pamela Denise McGee,01/12/1962,Flint,MI
Kayla Pedersen,04/04/1989,Flint,MI
```

If the output is based on a search, the search term should be listed as the heading. (Example: If the user asked for a partial match on name and entered "Jack" then the heading should be something like "`Partial Name Match for "Jack":`.")

Implementation Issues:

Text File I/O:

To perform output to a text file, use the **FileWriter** class with the **BufferedWriter** class as follows.

```
    FileWriter outfile = new FileWriter("output.txt");
    BufferedWriter bw = new BufferedWriter(outfile);
    bw.write("This is a test -- did it work?");
    bw.newLine();
    bw.close();
```

When you have finished writing to a file, you must remember to close it, or the file won't be saved. If you fail to close the file, it will be empty!

Remember to add '`throws IOException`' to the signature of any method that uses a **FileWriter** or

BufferedWriter or that directly or indirectly calls a method that performs File I/O.

Reading Input from the Keyboard:

In order to get information from the user for this project, you need to read input from the Keyboard. This can be done using the InputStream member of the System class, that is named "`in`". When this input stream is wrapped with a BufferedReader object, the `readLine()` method of the BufferedReader class can be used to read and store all of the characters typed by the user into a String. Note that `readLine()` will *block* until the user presses the Enter key, that is, the method call to `readLine()` will not return until the user presses the Enter key.

The following code shows how to wrap and read strings from `System.in` using an InputStreamReader and a BufferedReader.

```
BufferedReader inputReader = new BufferedReader(
                                    new InputStreamReader( System.in ) );
System.out.print( "Type some input here: " );
String input = inputReader.readLine();
System.out.println( "You typed: " +  input );
```

You need to add '`throws IOException`' to the signature of any method that uses or that directly or indirectly calls a method that uses a BufferedReader or InputStreamReader.

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 dates or ages or regular expressions or wild cards. Alternatively, think of ways to decompose one of the person class 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 **Wednesday, March 4th**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**.

The final version of your project including final design (UML, Javadoc, unit tests) and final implementation is due on **Wednesday, March 25th**. 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) copy of the assignment. This 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 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), and**

• 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.