

Project #2
Computer Science 2334
Fall 2013

User Request:

“Create a sortable and searchable scholarly publication data system.”

Milestones:

1. Use keyboard input to get information from the user. 5 points
 2. Use text file I/O to read and write text files. 10 points
 3. Create classes to store data on various types of scholarly publications, including journals, conference proceedings, and the papers published in each. Note that you should create any additional classes (abstract and/or concrete) and/or interfaces you deem necessary to arrive at a good design. 10 points
 4. Implement both the **Comparable** and **Comparator** interfaces to compare one scholarly paper to another. 10 points
 5. Use a **List** to store, retrieve, and display data related to scholarly papers as described below. 15 points
 6. Use the `sort()` and `binarySearch()` methods from the **Collections** class to sort and search for data related to the description below. 20 points
- ▶ Develop and use a proper design. (See Milestone 3, above.) 15 points
 - ▶ Use proper documentation and formatting. 15 points

Description:

Research is the creation of new knowledge. Research dissemination is the process of sharing with others the knowledge gained through research. Research dissemination typically takes place through the publication of scholarly works of various types. For this project, you will create a system to keep track of a few common types of scholarly publications. In particular, it will keep track of *papers* – brief peer reviewed works on a single topic, often describing original research – that are published in two types of collections of similar papers called *journals* and *conference proceedings*. Papers published in journals are called *articles* while those in conference proceedings are called *conference papers*. Journals and conference proceedings are *serials*. That is, journals and conference are published periodically. Conference proceedings are published in conjunction with conference meetings, which generally take place once per year but may be less frequent. Journals are generally published a few times per year.

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 scholarly publications. We will call this application *ScholarPub*. 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 name of the data file where publication data is stored. This should be done using the technique described in the section below on reading input from the keyboard. It will then read in the specified data file and store the data into ScholarPub. Each line in the data file describes a particular paper. The format of this file is described below under Input and Output Format.

Once the data is loaded from the file, your program will enter a loop where it asks the user for criteria on which to sort the data. As with the file name, this information should come from the keyboard using the technique described below. The possible sorting options the user can enter are 'BI' for bibliographic, 'AN' for author names, 'PT' for paper title, 'ST' for serial title, 'CH' for chronological, and 'R' for random. Alternately, the user may choose to enter 'PS' for print to screen, 'PF' to print to a file, or 'S' for search. If the user chooses either print option, then all of the publication data will be printed in its current order (whatever that may be, given the last sort option) and in the format described below under Input and Output Format. (If the print option is 'PF' the user will also be prompted for an output file name.) If the user chooses search, he or she will be prompted for the title of a paper on which to search, then your program will search for and display the data on that paper. (You may assume that the title is unique in ScholarPub.) If no paper with that title appears in the data searched, the user will be informed of that fact. The final option available to the user is 'E' for exit. If the user chooses exit, your program will thank him or her for running the program and exit without errors.

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 paper based on its title. If the data structure holding the paper 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 title, 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.

To observe this efficiency gain, you will measure the number of comparisons the system uses to find a paper based on title, given the ordering of the data in ScholarPub. Sort the data using one ordering (such as 'AN') then search for five different titles, record the values for the resulting number of comparisons, repeat this for each possible ordering of the data, then present them in a simple table using the format shown below. (You may need to adjust spacing for long titles.)

	1.	2.	3.	4.	5.
Title:					
Order	Search Time	Search Time	Search Time	Search Time	Search Time
-----	-----	-----	-----	-----	-----
BI					
AN					
PT					
ST					
CH					
R					

Note that some of the sort options are not related to the search terms that the user will be providing or do not define a unique ordering. For example, random (one of the possible sort options) is not related to the title of the paper searched for (the search term). Moreover, the ordering defined by this sort option is not unique. (Each time the random option is selected, the same list may be placed in a different order, so the ordering of the papers based on this criterion is arbitrary.) For these situations, you will have no choice but to search the collection linearly. On the other hand, when the sort option is related to the search

terms and does define a unique ordering, a binary search is preferred and should be used. For which sort option(s) is it appropriate to use a binary search (BI, AN, PT, ST, CH, or R)? *Explain your answers.* For which sort option(s) is it *not* appropriate to use a binary search? *Explain your answers.*

Put the table of data and your answers to these questions into milestones.txt under milestone 5.

Note that authors typically have multiple names. For ScholarPub, we will break each author's name into two parts – a primary name and a secondary name. The *primary name* is what we in western countries often call a “last name” or “family name.” This is the primary name in ScholarPub, since we will look first at the this name when sorting based on name. Only if the primary names are equivalent will the secondary name be considered (as a “tie-breaker”). The *secondary name* consists of whatever remains of the name beyond the primary name. This includes what we often call a “first name” or “given name” along with various “middle names.” So, for example, the full name “Dean Frederick Hougen” would have “Hougen” as the primary name and “Dean Frederick” as the secondary name. Note further that both primary and secondary names may have several parts to them separated by spaces.

Note also that there may be multiple authors on a given paper. This means that when sorting on author names, one should look first at the first author's names, then at the second author's names, and so forth until a difference is found. It is also possible for multiple papers to have the same author(s).

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

Input/Output Formats:

Input and Output Format

Each paper is described by several lines. The first line indicates the kind of paper. The second line is a list of authors. For each author the primary name is listed first, followed by a comma and a space then the secondary name. If there are multiple authors, a semicolon followed by a space separates one entry from the next. The third line is the paper title. The fourth line is the serial title. The fifth consists of volume (for journals only), issue (for journals only), and starting and ending page numbers. For journals, the issue number is in parentheses and the page numbers are preceded by a colon. The starting and ending page numbers are separated by a hyphen. The sixth line is publication month and year data. The seventh (not present for all entries) is a digital object identifier.

Examples:

Conference Paper

Woehrer, Mark; Hougen, Dean; Schlupp, Ingo

Sexual Selection, Resource Distribution, and Population Size in Synthetic Sympatric Speciation
International Conference on the Synthesis and Simulation of Living Systems

137-144

July 2012

<http://dx.doi.org/10.7551/978-0-262-31050-5-ch020>

Journal Article

Eskridge, Brent E.; Hougen, Dean F.

Extending Adaptive Fuzzy Behavior Hierarchies to Multiple Levels of Composite Behaviors
Robotics and Autonomous Systems

58(9):1076-1084

September 2010

Implementation Issues:

File I/O:

To perform output to a 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 Be sure to use UML class diagrams as tools to help you with the design process.

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, 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.

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 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, and 8.

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

11 Test the overall program and fix any bugs.

12 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.

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 publication year or individual terms from the paper title or regular expression or wild cards. Alternatively, think of ways to decompose

the class for paper data 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 paper data from the text file.

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:

The electronic copy of your preliminary design (UML, stub code, detailed Javadoc documentation, and unit tests) is due on **Wednesday, September 25th**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**. Submit your UML design on *engineering paper* or a hardcopy using UML layout software at the **beginning of lab on Thursday, September 26th**.

The electronic copy of the final version of the project is due on **Wednesday, October 2nd**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**. Submit your final UML design on *engineering paper* or a hardcopy using UML layout software at the **beginning of lab on Thursday, October 3rd**.

You are not allowed to use the **StringTokenizer** class. Instead you must use `String.split()` and a regular expression that specifies the delimiters you wish to use to “tokenize” or split each line of the file.

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 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.