# Project 2

*Computer Science 2334*

*Spring 2017*

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

## *User Request:*

*"Create a sortable and searchable data system for newspaper and TV news stories."*

## *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 newspaper and TV news stories. 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 newspaper or TV news story to another. *10 points*
5. Use a **List** to store, retrieve, and display data related to newspaper and TV news stories as described below. *15 points*
6. Use the `sort()` and `binarySearch()` methods from the **Collections** class to sort and search for data related to newspaper and TV news stories as described below. *20 points*

► Develop and use a proper design. (See Milestone 3, above.) *15 points*

► Use proper documentation and formatting. *15 points*

## *Description:*

Newspaper stories have certain basic features, such as publication date, word count, and news maker covered. Similarly, TV news stories have basic features such as broadcast date and news maker covered. However, word counts are not kept for TV news stories. Instead, each TV news story's broadcast duration (in seconds) is recorded. For this project, you will create a system that models both newspaper and TV news stories.

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 newspaper and TV news stories. As with Project 1, we will call this application *Nooz*., except that this will be *Nooz 2.0* (in contrast to what we will retrospectively consider the *Nooz 1.0* of Project 1.) 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.

Nooz 2.0 will first ask the user for a file in which news data is stored. This should be done using the technique described in the section below entitled Reading Input from the Keyboard. Nooz 2.0 will then read in the specified data file and store the data. Each line in this data file describes a particular news story. The format of this file is similar to that for Project 1, except that some lines will describe newspaper stories and others will describe TV news stories. This requires a slightly more complex data line for each news story. The exact data file format is described below in the section entitled Input Format. Also, while Nooz 1.0 did not consider the possibility that a given story was already found in the database, Nooz 2.0 should consider this possibility.

Once the data is loaded, Nooz 2.0 will enter a loop where it asks the user questions. As with the file name, 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, Nooz 2.0 will proceed through a series 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 "Search newspapers, TV news, or both (n, t, or b)?" The second question is "Search news makers by exact or partial matches (e or p)?" Depending on the answer to the second question, the third question will either be "Newsmaker (exact)?" or "Newsmaker (partial)?" as appropriate. Next, Nooz 2.0 will ask, "Primary sort criterion is source, topic, or length (s, t, or l)?" Finally, Nooz 2.0 will ask, "Secondary sort criterion is … ?" Here the ellipsis will be replaced with the other two possible sort criteria (that is, not the one selected as the primary sort criterion).

For most of the questions, Nooz 2.0 should only accept the one letter responses indicated as possibilities within parentheses for each question. For "Newsmaker?" any string is a valid response (but may not match with any news maker in the data). If the user enters no text, just a return, for news maker, the special news maker "None" will be assumed.

In all cases, once the user has answered all questions, 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 (y/n)?" If the user chooses "y," Nooz 2.0 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, Nooz 2.0 will ask the user "Continue (y/n)?" If the user's response to continue is "y," Nooz 2.0 will return to the first question ("Search newspapers, TV news, or both (n, t, or b)?"). If the user's response to continue is "n," Nooz 2.0 will thank the user for using Nooz and exit gracefully. ("Thanks for using Nooz.")

If the user response to any question is incorrect (that is, does not allow Nooz to proceed to the next step), Nooz should present the user with a polite message stating that and asking the user to give new input ("The choice you entered is invalid. Please enter a new choice.").

## *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. In Nooz 2.0 we will use sorting for both purposes.

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 doing exact searches for news makers. (Why only for exact searches? Answer this question along with your noun/verb/adjective list.) Indeed, once the data is read in, you should keep the list of news makers sorted by name.

While it makes sense to keep the news makers sorted by name, the user may want the list of stories displayed to be sorted one way for one query and another way for another query. We have given the user the choice to sort the stories by source (name of newspaper or TV news show), topic, or length. However, note that a collection in Java can have at most one *natural ordering*. We will consider topic to be the most appropriate natural ordering for news stories and we will implement **Comparable** and define the `compareTo()` method(s) to use that ordering. The other sort options will need to be implemented

using the `compare()` method that comes from implementing **Comparator**. Note that for comparing lengths, we'll use a speaking rate of 150 words per minute to convert duration in seconds to approximate word count.

### *Input/Output Formats:*

Input Format

The input file format for newspaper and TV news stories is similar to the input file format for Project 1. Each data line contains the date the story was published or broadcast, followed by the source of the story (either a code for a newspaper as in Project 1 or a code for a TV show as described below), followed by the story length (a count of words in the story for newspaper stories or a duration in seconds for TV news stories), followed by a topic for the story, followed by the lead newsmaker in the story, followed by the second lead newsmaker in the story. These data fields are all separated from one another by commas and each newsmaker name is surrounded by quotation marks. The codes for topics and newspaper names are given in Project 1 and the special code 99 (for "None") is retained from that project. The TV news show codes are as given below.

| | |
|---|---|
| 401 | ABC Good Morning America |
| 402 | ABC World News Tonight |
| 421 | NBC Today |
| 422 | NBC Nightly News |
| 441 | CBS The Early Show |
| 442 | CBS Evening News |
| 461 | PBS NewsHour, 1st half hour |
| 462 | PBS NewsHour, 2nd half hour |
| 500 | CNN unspecified show |
| 501 | CNN Daytime |
| 502 | CNN Situation Room |
| 504 | CNN Anderson Cooper 360 |
| 511 | CNN John King, USA |
| 514 | CNN Erin Burnett Outfront |
| 520 | MSNBC unspecified show |
| 521 | MSNBC Daytime |
| 522 | MSNBC Hardball with Chris Matthews |
| 528 | MSNBC Rachel Maddow Show |
| 530 | MSNBC The Ed Show |
| 532 | MSNBC PoliticsNation |
| 540 | Fox News unspecified show |
| 541 | Fox News Daytime |
| 543 | Fox News O'Reilly Factor |
| 545 | Fox News Fox Report with Shepard Smith |
| 547 | Fox News Hannity |
| 548 | Fox News Special Report with Bret Baier |

Output Format

The output format for each line of data will be the same as described for Project 1 with the exception that for TV news shows, the length will be given in seconds rather than words. The order of the output lines will be based on the choices the user made for primary and secondary sort criteria. (The sort criterion not selected will be the tertiary sort criterion by default.)

As with Nooz 1.0, after the individual data lines, Nooz 2.0 will present a summary line, as follows. If the user only searched newspaper stories, the summary line will list the number of stories found, the number of different newspapers in which these stories were published, the total number of words in these articles, and the number of different topics found. If the user only searched TV news stories, the summary line will list the number of stories found, the number of different TV news shows in which these stories were broadcast, the total number of seconds in these stories, and the number of different topics found. Finally, if the user searched either newspaper or TV news stories, the summary line will list the number of stories found, the number of different sources (newspapers or TV news stories) for these stories, the total number of word equivalents in these stories (converting from seconds to words at a rate of 150 words per minute), and the number of different topics found.

### *Implementation Issues:*

File I/O:

To perform output to a file, use the **FileWriter** class with the **BufferedWriter** class. An example follows. You will, of course, need to modify this example to use the name of the file specified by the user and to output the data in the format described above.

```
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 Before, during, and after the lab session, you should 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 fields 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 system 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/fields, methods, and interfaces (if any).

1.5 Next, use UML class diagrams as tools to help you establish proper relationships between your classes, fields, 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), field 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 At the end of the first week, you will turn in your preliminary design documents (see ***Due Dates and Notes***, below), which the TAs will grade. **There will be no late work accepted for the 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

5 After the design due date, you will be provided with a UML design that you must follow for your completed project.

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

7 Create a new set of Javadoc files using Eclipse and inspect them to make sure they're appropriate.

8 Implement the design you have developed by coding each method you have defined.

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

10 Test the overall program and fix any bugs.

11 Submit your project (see ***Due Dates and Notes,*** below).

### *Due Dates and Notes:*

Your preliminary design (list of nouns, verbs, and adjectives; UML; stub code; detailed Javadoc documentation; and unit tests) is due on **Friday, 17 February 2017**. Submit the project archive following the steps given in the submission instructions **by 11:00 pm**.

The final version of your project is due on **Monday, 27 February 2017**. 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.