Project 1

Computer Science 2334
Spring 2017

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

User Request:

"Create a simple newspaper story data system."

Milestones:

1. Use program arguments to specify a file name. *10 points*
2. Use simple File I/O to read a file. *10 points*
3. Create an abstract data type (ADT) to store information on a single newspaper story. *15 points*
4. Create an ADT that *abstracts* the use of an array (or list) of newspaper stories. *15 points*
5. Implement a program that allows the user to search the list of newspaper stories as described below. *20 points*

- ► Develop and use a proper design. *15 points*
- ► Use proper documentation and formatting. *15 points*

Description:

For this project, you will put together several techniques and concepts you have learned in CS 1323 (or from a similar background) and some new techniques to make an application that searches a collection of data on newspaper stories. This application will be called *Nooz* and will allow users to enter the names of newsmakers (that is, the people the newspaper stories are about) and see basic data (such as publication date) of each relevant newspaper story according to the data in the database.

One of the positive aspects of this project is that it will use an arbitrary amount of data. Nooz must be capable of handling data on hundreds or thousands of newspaper stories. To the surprise of no one, the best approach to this problem is to decompose the problem into separate classes that can be gradually built up. Note that much of the code you write for this program could be reused in more complex applications, which, by the way, is what we will do in our projects this semester.

Operational Issues:

Data file and format:
Nooz will read the data file (a text file) as specified by a file name. The file name will be given as a program argument. (See ***Implementation Issues*** below or refer back to your first lab for information on how to read program arguments). The first line of the file contains header information. Each line of the file, except the first, contains information on a single newspaper story. Each data line contains the date the story was published, followed by the newspaper in which the story was published, followed by a count of words in the story, 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 date is given as a single number, the first four digits of which represent the year, the second two the month, and the final two the day. The newspaper is encoded as a single number, with the following possible values.

Code	Newpaper
1	New York Times
2	Washington Post
3	Wall Street Journal
4	USA Today
13	Los Angeles Times
117	Denver Post
118	Houston Chronicle
119	Orlando Sentinel
120	Traverse City Record
121	Daily Herald (Everett, WA)
122	Eagle Tribune (MA)

The topic of the story is likewise encoded as a single number, with the following possible values.

Code	Topic
1	Government Agencies/Legislatures
2	Campaigns/Elections/Politics
3	Defense/Military (Domestic)
4	Court/Legal System
5	Crime
6	Domestic Terrorism
7	Business
8	Economy/Economics
9	Environment
10	Development/Sprawl
11	Transportation
12	Education
13	Religion
14	Health/Medicine
15	Science and Technology
16	Race/Gender/Gay Issues
17	Immigration
18	Additional Domestic Affairs
19	Disasters/Accidents
20	Celebrity/Entertainment
21	Lifestyle
22	Sports
23	Media
24	U.S. Miscellaneous
25	U.S. Foreign Affairs
26	Foreign (non-U.S.)

For example:

```
20120102,1,1374,2,"Santorum, Rick","Romney, Mitt"
```

This indicates that the story was published on January 2, 2012 in the New York Times, that it was 1374 words long, that the topic was Campaigns/Elections/Politics and that lead newsmakers of the story were Rick Santorum and Mitt Romney. Note that besides the name of an individual, a newsmaker may be an

organization such as "Obama Administration" or, if there are not two newsmakers for a story, either or both may be replaced by the numeric code 99. For example:

```
20120403,1,2973,1,99,"Obama Administration"
```

This indicates that the story was published April 3, 2012 in the New York Times, that it was 2973 words long, that the topic was Government Agencies/Legislatures and that the one lead newsmaker of the story was the Obama administration.

You will need to store the data on each newspaper story as an object and the collection of all newspaper stories will be stored as a list of these objects. In addition, you may create and use objects of other types to give your system a logical design and the functionality required by the program specifications.

User interaction with Nooz:
Once the list of newspaper stories has been read into Nooz and stored, Nooz will use a **JOptionPane** to display to the user a dialog box requesting the name of a newsmaker.

When the user enters a newsmaker name into the dialog, Nooz will check to see if the name is associated with any newspaper story in the database. If so, Nooz will use another dialog to display to the user the data of all of the newspaper stories in the database associated with that name. In particular, Nooz will show the following information for each associated story: Date published (in the format month day, year, where month is spelled out as a word), newspaper name (in words, not the numeric code), word count, and topic, all separated by semicolons. For example, if the user entered the name "Santorum, Rick" in the input dialog, one line of the response, corresponding to the data above would be as follows:

```
January 2, 2012; New York Times; 1374 words; Campaigns/Elections/Politics
```

One such line will be displayed for each individual newspaper story found. After the lines for all of the individual newspaper stories are displayed, Nooz will also give one summary line that lists 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 newsmaker name is *not* associated with any newspaper story in the list, Nooz will use a dialog to inform the user of that fact.

Note that the user may also leave the newsmaker name field empty on the input dialog. If the user does so, Nooz should find all stories for which no newsmaker is given (i.e., those for which both newsmaker data fields in the file are the special code of 99) and print individual lines for those stories as well as a final summary line.

After checking whether the name is associated with any story in the database and displaying information to the user one way or the other, Nooz will again use a dialog to request another newsmaker name. It will continue in this loop until the user clicks on cancel, at which time Nooz should gracefully exit.

Implementation Issues:

There are two Java elements in this project that may be new to some students: reading from a file and (if you skipped Lab 1) program arguments. These Java features are summarized below.

Reading from a file:

We will discuss File I/O in more depth later in the class; this project is just designed to give you a brief introduction to the technique. Reading files is accomplished in Java using a collection of classes in the **java.io** package. To use the classes you must import the following package:

```
import java.io.IOException;
```

The first action is to open the file. This associates a variable with the name of the file sitting on the disk.

```
String fileName = "StoryData.csv";
FileReader fr = new FileReader(fileName);
```

(Note that the lines given above will work if your data file is called "`StoryData.csv`." However, you should *not* "hardcode" this file name into your source code. Instead, you should get the name of the file from a program argument when your program is run. You will, therefore, need to modify the code provided above to use the variable in which you have stored the program argument.)

Next the **FileReader** is wrapped with a **BufferedReader**. A **BufferedReader** is more efficient than a **FileReader** for working with groups of characters (as opposed to individual characters). Another advantage of using a **BufferedReader** is that there is a command to read an entire line of the file, instead of a single character at a time. This feature comes in particularly handy for this project.

```
BufferedReader br = new BufferedReader(fr);
```

The **BufferedReader** can now read in Strings.

```
String nextline;
nextline = br.readLine();
```

Look at the Java API listing for **BufferedReader** and find out what `readLine()` returns when it reaches the end of the file (stream). Have your code process each line, putting the data into objects and variables while also looking for this special return value. When you are finished with the **BufferedReader**, the file should be closed. This informs the operating system that you're finished using the file.

```
br.close();
```

Closing the **BufferedReader** also closes the **FileReader**.

Any method which performs I/O will have to throw or catch an **IOException**. If it is not caught, then it must be passed to the the calling method. The syntax is given below:

```
public void myMethod(int argument) throws IOException {
        //method body here
}
```

Program Arguments:

Sometimes it is handy to be able to give a program some input when it first starts executing. Program arguments can fulfill this need. Program arguments in Eclipse are equivalent to MS-DOS, Mac, or Unix command line arguments. Program arguments are handled in Java using a **String** array that is traditionally called `args` (the name is actually irrelevant). See the "Lab 2" slides (this year provided for Lab 1) for how to supply program arguments in Eclipse.

The program below will print out the program arguments.

```
public static void main(String[] args) {
        System.out.println(args.length + " program arguments:");
        for (int i=0; i< args.length; i++)
                System.out.println("args[" + i + "] = " + args[i]);
}
```

(Note that your program should not print the arguments but, instead, use the appropriate argument as the filename from which to read the data.)

Milestones:

A milestone is a "significant point in development." Milestones serve as guides in the development of your project. Listed below are a set of milestones for this project along with a brief description of each.

Milestone 1. Use program arguments to specify a file name.
The name of the file that stores the list of data on newspaper stories will be passed to the program using program arguments as discussed above. Type in the sample program given in the section on program arguments and make sure that you understand how the program arguments you provide affect the `String[] args` parameter that is passed into the `main` method of the program. Then, write a `main` method for your program that reads in the name of the data file from the program arguments.

Milestone 2. Use simple File I/O to read a file.
Before you can allow the user to search the list of newspaper stories, you must first be able to read a text file. Examine the section above on reading from a file. A good start to the program is to be able to read in the name of a file from the program arguments, read each line from the file, one at a time, and print each line to the console using `System.out.println()`. Later, you will want to remove the code that prints out each line read in from the file, since the project requirements do not specify that the file is to be written out to the console as it is read.

Milestone 3. Create an abstract data type (ADT) to store data on a single newspaper story.
You must create an ADT that holds the data for a single newspaper story from the data file before you can store that data. Think about what data is associated with each newspaper story and how to most efficiently store and retrieve the data. Also, think about any methods that may help you to manage and compare the data by abstracting operations to be performed on individual entries in the list. Such methods may be used by other classes.

Milestone 4. Create an ADT that abstracts the use of a list of data about newspaper stories.
You are to store the object representing each newspaper story into a list of objects. However, it is not necessary for the portions of the program that will carry out user actions to directly operate on this list as they would if you simply used an array of newspaper story objects. Instead, you should create a class that abstracts and encapsulates this list and allows for the addition of new newspaper stories and also supports other required operations on it.

This ADT will represent the collection of information associated with the program. Think about the operations that this ADT needs to support and how it will use the ADT created for Milestone 3. At this point, you should be able to read in the input file and create an object for each newspaper story in the file, and store that object in the list. Note that the data file used for grading may be larger (or smaller) than the data file provided for testing.

Milestone 5. Implement a program that allows the user to search the list of newspaper stories.
This is where the entire program starts to take on its final form and come together. Here you will create the input and output dialogs and the menu system. Start by creating the input dialogs and the output dialogs. Tie together the input dialogs, the ADT from Milestone 4, and the output dialogs to make this search functional and test its functionality.

Finally, you are ready to create the main loop of the program that will take input and invoke the correct methods to create appropriate output.

Remember that when the user clicks on "cancel," the program must gracefully exit. This can be accomplished by using `System.exit(0)`.

How to Complete this Project:

Preliminary Design:

1 During the lab session and in the week following, 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 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 and methods 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. 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 the classes until after the design is completed.

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 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 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:

Due dates:
Your preliminary design (list of nouns, verbs, and adjectives; UML; stub code; detailed Javadoc documentation) is due on **Friday, February 3rd**. Submit the project following the steps given in the submission instructions **by 11:00pm**.

The final version of the project is due on **Monday, February 13th**. Submit the project following the steps given in the submission instructions **by 11:00pm**.

Academic honesty:
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.

ADTs:
Do not be confused by the term "abstract data type" (ADT). An ADT is **not** the same as an abstract class, even though they both contain the word "abstract" in them.

A data type is simply a description of how bits in a computer are grouped and interpreted. Maybe one set of 32 bits is interpreted as a character, whereas another set of 32 bits is interpreted as an integer, and a set of 64 other bits is interpreted as an integer that can hold larger magnitudes, etc. With *concrete* data types, implementation details matter, such as the number of bits, whether the bits are ordered from least to most significant, etc. If you try to mix implementations, you'll screw things up.

With an abstract data type, you hide the implementation details of the data type from the user, so that what matters is how one *interacts* with instances of the type, not how they are *implemented* internally. So, if you add two integers whose internal representations differ, you should still get a sensible result.

This means that if you create a class using object-oriented techniques (such as making variables private and only accessible through methods, etc.), then even a *concrete* class is an *abstract* data type.

The reason this description doesn't just tell you to create a class to store each newspaper story's data is because you don't have to use just one class. You could use two classes, or three, or more. You could arrange them in an inheritance hierarchy (where one is a subclass of another). You could use composition or aggregation (the types of has-a links we have discussed). All of these classes could be concrete or some of them could be concrete and some could be abstract. You could also include interfaces, if you saw a good reason to do so. All of these alternatives would count as creating an ADT. However, you should also strive for simplicity; don't make an inheritance hierarchy or a bunch of classes or interfaces just because you can—try to match your design to the requirements.

If the term 'ADT' is still confusing you, think of the assignment as saying "Create something appropriate that the computer can use to store each newspaper story's data." That is what it means.

Hint:
Consider using an **ArrayList** in conjunction with Milestone 4.