CECS 277 HOMEWORK MEMENTO

OBJECTIVE: Learn how to use the Memento design pattern and get practice writing binary objects out to a file.

INTRODUCTION: Your text illustrates how to write multiple objects out to a binary file using stream I/O. In the text, those objects are arranged in some collection class, say an array or ArrayList, and your code writes out the whole array or ArrayList to the file in one statement. This approach will not scale well if you get a large collection of large objects. It is better to store instances of a given object out to a file one at a time if you do not know for sure how many you will eventually need to write.

In many of the articles online regarding the Memento pattern, the state shows up as a String. The authors do that to make the example code simpler, and hopefully easier to understand. But you have to remember that generally speaking, the state of an object is the combination of the value of all of its instance variables. The instances of a large complex class could each have a very complex state indeed.

For this homework assignment, we are going to simulate a somewhat contrived personnel records application. Each time that you run your application it will store multiple sets of values for the same person. In our simplified world, their first and last name never changes, but their hair color, height, and weight may change from one set of "readings" to the next. For instance, I have been 5'10" for quite some time, and I have not started to lose height yet. So my height is pretty stable. But my hair has gone from auburn to gray, to salt and pepper to nearly white. During that same span of time, my weight has gone up and down some as well. So there will be multiple instances of Person for me, each one reflecting my statistics at a given time.

In this homework assignment the Caretaker writes each memento to your binary output stream **individually**, rather than all at once. The `MementoRunner` class will prompt the user for a fresh set of values for the Person (more on that in a minute) and make a `PersonMemento` instance from that Person instance and then store that `PersonMemento` in your stream object file.

Then, once all of the `PersonMementos` have been written to disk, you will read all of those `PersonMemento` instances from start to finish, and find the one which has the lowest weight. Then, in a separate statement, I want you to prompt the user for a weight to search for, then find the Memento whose corresponding Person instance has that weight and return the first Memento that you find with that weight.

PROCEDURE:

1. Start with the Person class. This class performs the role of the Originator in the Memento design pattern. The code for that is here.
    a. You need to add a `save()` method that creates an instance of `PersonMemento` (more on that in a minute) from the current `Person` instance.

   b. You need to add a `restore (PersonMemento archive)` method that returns an instance of Person that has the same state as the Person instance stored in the supplied `PersonMemento`.
2. Then write your PersonMemento class, which performs the duties of the Memento class in the design pattern:
   a. A constructor that takes just one argument: the Person instance that you want to preserve into an instance of PersonMemento.
   b. A `getSavedPerson` method that returns an instance of `Person` with the same values as the Memento.
   c. Remember that your `Caretaker` class will write instances of `PersonMemento` out to a file output stream file, so that means that you must implement the `serializable` interface.
3. Then write your Caretaker class:
   a. The `Caretaker` class does all of the I/O with your Memento file that has all of your Memento instances (not Person instances) in it. Once you establish an instance of `Caretaker`, that object has the same file that it writes to and reads from.
      i. The caretaker class could just randomly come up with a file name for the mementos written out to disk, but its safer to prompt the user for that file name.
      ii. Caretaker will have a single constructor that takes in a file name for the binary file that it uses to write to/read from.
         1. You prompt the user for the name of that file in your runner (Main) method.
         2. The caretaker will need to first create a FileOutputStream from that file name, then use that FileOutputStream to create a new ObjectOutput Stream.
         3. The caretaker will also need to create a new FileInputStream that points to the same binary file as the FileOutputStream, and an ObjectInputStream from that FileInputStream.
         4. Both the FileInputStream and FileOutputStream need to be member variables in your caretaker instance.
         5. The addMemento will always append the new memento to the end of the binary file that the caretaker "owns".
         6. Each time the caretaker performs a getMemento, you will need to:
            a. Close the binary file that you have been writing your Mementos to.
            ~~a.~~b. Create a new ObjectInputStream pointing to the binary file so that you can start at the front of the file again when you start reading in mementos.
   b. Write your `addMemento` method to write out a Memento instance.
   c. Write two methods to search through a file of Mementos. ~~Both of these~~Both methods are called `getMemento`:
      i. The first one has no arguments, and simply returns an instance of `PersonMemento` that has the lowest weight.

   ii. The second `getMemento` method takes an integer weight as its only argument, and returns the first instance of `PersonMemento` that it finds in the file that has a weight that matches the input weight. If **none** of the Mementos have exactly the weight requested, then return a `null` pointer.

4. Finally, write a driver class that:
   a. Prompts the user for the name and path of the file that they will use for storing their PersonMemento objects.
   b. Goes into a loop, prompting the user for the Person characteristics (height, weight, …) at various times.
   c. Uses getMemento to find the skinniest of the PersonMementos that have been written to disk.
   d. Uses getMemento to find the first PersonMemento that matches the weight that you give it.

SAMPLE OUTPUT:

Note that I used JFileChooser to prompt the user for the file name.

```
prompting you for persons:
Enter the person's last name: Berry
enter the person's first name: Blue
0:    BLACK
1:    BLONDE
2:    RED
3:    AUBURN
4:    SALT_AND_PEPPER
5:    GREY
6:    WHITE
7:    BALD
Please enter the # corresponding to the hair color: 1
You entered color: BLONDE
Person's height in feet: 5
And the inches?  For instance, 5'10" would be 10: 11
Their weight in pounds: 132
Name: Berry, Blue, Hair Color: BLONDE, Height:5'11, Weight #: 132
Are we done here (Y/N)?: N
0:    BLACK
1:    BLONDE
2:    RED
3:    AUBURN
4:    SALT_AND_PEPPER
5:    GREY
6:    WHITE
7:    BALD
Please enter the # corresponding to the hair color: 5
You entered color: GREY
Person's height in feet: 5
And the inches?  For instance, 5'10" would be 10: 10
Their weight in pounds: 136
Name: Berry, Blue, Hair Color: GREY, Height:5'10, Weight #: 136
Are we done here (Y/N)?: N
0:    BLACK
1:    BLONDE
```

```
2:     RED
3:     AUBURN
4:     SALT_AND_PEPPER
5:     GREY
6:     WHITE
7:     BALD
Please enter the # corresponding to the hair color: 6
You entered color: WHITE
Person's height in feet: 5
And the inches?  For instance, 5'10" would be 10: 9
Their weight in pounds: 150
Name: Berry, Blue, Hair Color: WHITE, Height:5'9, Weight #: 150
Are we done here (Y/N)?: Y
Skinniest version: Name: Berry, Blue, Hair Color: BLONDE, Height:5'11, Weight
#: 132
What weight do you want to search for?136
Sought after version: Name: Berry, Blue, Hair Color: GREY, Height:5'10,
Weight #: 136
Completed satisfactorily.
```

WHAT TO TURN IN:

- MementoRunner.java
- Person.java
- PersonCaretaker.java
- PersonMemento.java
- console.txt
- The UML diagram class for your project.