Version 1.0 : Released April 1st

Assignment 3

Company Management Tool Prototype (Part 3)

Due: not later than Sunday, April 14th, 2019, 11:59 p.m.

Description:

In this assignment you'll add additional features to the Company Management Tool application, as described in this document. In the process you will demonstrate your understanding of the following course learning requirements, as listed in the *CST8284—Object Oriented Programming (Java)* course outline:

1. Write java programming code to solve a problem, based on the problem context, including UML diagrams, using object-oriented techniques (CLR II)
2. Write Java code to implement exception handling (CLR VI)
3. Implement program Input/Output operations (CLR VII)
4. Produce tested code that executes correctly and consistently (CLR VIII)
5. Prepare program documentation using prescribed program specifiers, including JavaDoc (CLR IX)
6. Debug program problems using manual methods and computerized tools in an appropriate manner (CLR X)
7. Identify appropriate strategies for solving a problem (CLR XI)

Worth
5%
of your total mark

Assignment 3

Company Management Tool Prototype (Part 3)

I. Create a new project folder

a) In Eclipse, open your existing Assignment 2 folder, right click on the `src` folder, and select 'Copy' from the drop down menu.

b) Create a new project called CST8284_W19_Assignment3. Right click on the project folder, and select 'Paste' from the drop down menu. This effectively copies all the files in the Assignment2 `src` folder into the new assignment folder. Refactor package and class names as necessary.

II. Add the following new features to the Company Management Tool Prototype application

a) **Add File I/O capability**

Using the hybrid and lecture notes as your guide, add file I/O functionality to your application. You'll need to create two methods in Company, callable from CompanyConsole,, named:

`saveEmployeesToFile()`

Using getEmployees() to retrieve a handle to the employee ArrayList, load each Employee object in the ArrayList into a file called `CurrentEmployees.emp`. DO NOT hardcode the file location to a particular subdirectory; your code will not work correctly when it is transferred to another device. Also, do not prompt the user for a filename; this file is used internally by the program. Simply use the default directory associated with you project (which is typically the `src` folder or one of its subdirectories) to store the file.

`loadEmployeesFromFile()`

This method performs the opposite operation, loading the file contents into the employee ArrayList. As indicated in the lecture notes, you should use EOFException to terminate loading the ArrayList.

Students often encounter problems loading and unloading the files initially. The most common cause is that the file you think you are loading from/to does not actually exist. Check carefully to ensure that there's actually a file where you think it is (using debug, of course.)

Remember that you code needs to check for the existence of the file first, and if it doesn't exist, needs to create it. Students frequently assume that because their program works fine with the file loaded on their laptop, it will work fine everywhere. But when you lab instructor runs your program, there's no guarantee that the same file will be available on his/her laptop. If this happens, you've lost a number of marks, because a major component of the assignment would not execute at all. So test you code thoroughly.

Another potential complication is serialization. To ensure this doesn't cause you problems, add the following line to the Employee class

```
public static final long
                serialVersionUID = 1L;
```

Additionally, in additional to the EOFException mentioned above, you'll need to add FileNotFoundException.

You need to be very careful when testing the latter exception. Specifically, you're never going to trigger this exception if the file is always there. So you need to delete this file to test that your code recovers

correctly when `CurrentEmployees.emp` is not available.

b) **Modify the menu accordingly**

In the CompanyConsole class, add two new static final ints:

```
private static final int
        SAVE_EMPLOYEES_TO_FILE = 6;
private static final int
      LOAD_EMPLOYEES_FROM_FILE = 7;
```

and use these in the menu() method to call the two methods introduced in part (a) above.

c) **Add exception handling to your code**

Add exception handling to the Employee and OurDate classes to ensure that only correctly-formatted data can be instantiated.

Feel free to trap any exceptions that help improve the stability of your code. This includes adding a new Exception subclass to your project if this helps the readability and execution of the code.

Additionally, recall that we set the `isMaximumEmployees()` method to return false in Assignment 2. You should now modify this to catch an `OutOfMemoryException` and use this to determine if more employees can be stored in the heap space.

Note that your code will be tested with input typically input by users—which is to say, characters where numbers were expected, numbers where characters are expected, empty or null strings, etc. Your code needs to be robust enough to handle these possibilities. At the same time, it's important that the code not become cluttered with an excessive number of try…catch… blocks. So use judgement in determining how, where, and how many exceptions to use.

d) **Document your code using JavaDoc**

JavaDoc adds hypertext-linked documentation to your code, allowing you to output high quality comments similar to those seen on the Oracle web site for each class. Provided, that is, you take the time to lay out your comments correctly. (See the sample documentation, available at the Oracle web site, on the next page.)

A good place to begin with JavaDoc is

https://idratherbewriting.com/java-javadoc-tags/

which explains what each tag does. For this assignment, the following tags are essential:

For the class:

@author
@version

For methods and constructors

@param
@returns
@throws

You can safely ignore @see and @since for this assignment. Feel free to other JavaDoc and HTML tags as you find are appropriate.

You'll also want to refer the *Algonquin College Documentation Standard* (ACDS), a series of (somewhat dated) slides indicating which components of your code need to be documented (short answer: pretty much everything). The document is available on Brightspace. The ACDS specifies that you must document both the file and the class. (This is because a file may contain more than one class.)

A doc comment is written in HTML and must precede a class, field, constructor or method declaration. It is made up of two parts -- a description followed by block tags. In this example, the block tags are @param, @return, and @see.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
}
```

From: "How to write Doc Comments for the JavaDoc Tool", Oracle Corp., downloaded from http://www.oracle.com/technetwork/articles/java/index-137868.html April 11, 2017

In this assignment, you're fee to ignore that requirement: document the class only.

Despite your efforts to document your code correctly, it will all be for nothing if you don't follow the instructions in the Appendix *exactly*. This is because the JavaDoc utility does not operates on all the folders in your project, but *only* on the folders and subfolders that you select initially, before running the utility. So if you click on a *class* and then execute the JavaDoc utility, you will only generate JavaDocs for the methods and constructors in that folder. You must be careful to select the *project first*, and only then run the JavaDoc utility, according to the instruction below, if you plan to get full marks for this part of the assignment.

III. Notes, Suggestions, and Warnings

a) Before contacting the instructor about problems you are having with your code, be sure to first use debug to isolate the location of the problem. This involves setting a breakpoint in your code to just before the 'last known good' location (as indicated in the red output message that appear in the console), and step until the error is encountered, using the Variables windows to check the state of the fields at various stages. (And if the 'last know good' location is the first line of the program, then that's where you'll set your breakpoint.) Fix the bugs as they appear, run the code again, and repeat until there are no 'red' error messages. Then check for runtime errors by rigorously testing your working code.

Contact your lab professor if you are stuck, but only after you've made a valid attempt to fix your program using debug.

b) Students are reminded that:
   - You may use the Assignment 1 code as your basis for building Assignment 3, either in whole or in part. You do not need to cite the professor/author by name when using this code.
   - You should not need to use code/concepts that lie outside of the ideas presented in the course notes, labs, or hybrids. If you *do* employ concepts in your assignment that are not covered in the course, you *must* cite the source of this information. Failure to do so *will* result in a charge of plagiarism.
   - Students must be able to explain the execution of their code. If they can't, then it is reasonable to question whether they actually wrote the code or not. Partial marks, including a mark of zero, may be awarded if a student is unable to explain the execution of their code, and a charge of plagiarism may be levelled if there is good reason to believe the student did not author the code they submitted.

c) New versions of this document will appear as bugs are uncovered, or as further clarification is required regarding certain questions that arise. Check Brightspace periodically for revisions.

IV. Submission Guidelines

To submit your project, in Eclipse, right click on the project name (the project, NOT the package name), select Export... In the Export dialog, select General >> Archive File, then click the Next button. In the Archive File menu, (making sure your Assignment1 project is selected, along with all its components, at right), in the text box, name your zip file according to the naming convention indicated below.

Save this zip file to any location on a drive where you can find it, and select Finish. This is the zip file you should upload.

The correct format for labelling your zipped project is:

```
Lastname_Firstname_Assignment3_SecXXX.zip
```

where Lastname and Firstname are, of course, *your* last name and first name, and XXX is your lab section number. (Note the underscores, the use of upper and lower case, and no spaces; this format must be followed exactly when you name your archived file.) Lab times and section numbers are:

Tuesday 8:00 a.m. – 10:00 a.m.: Sec 301
Tuesday 11:00 a.m. – 1:00 p.m.: Sec. 312
Tuesday 1:00 p.m. – 3:00 p.m.: Sec. 303
Tuesday 3:00 p.m. – 5:00 p.m.: Sec. 313
Tuesday 6:00 p.m. – 8:00 p.m.: Sec. 314
Friday 10:00 a.m. – 12:00 a.m.: Sec. 302
Friday 2:00 p.m. – 4:00 p.m.: Sec. 304
Friday 4:00 p.m. – 6:00 p.m.: Sec. 311

Failure to label your submission according to these rules will cost you marks.

Note:

- *Since there are always some students who upload empty zip files with their submission, it is highly advisable that before you upload your final submission to Brightspace, you copy it to another location first and unzip it, making sure it contains the correct information. If you've followed the instructions in this document correctly, when you open your zip, you should see the CST8284_W19_Assignment3 folder, and inside that, appropriate subfolders,*

including `src` *and* `doc` *folder. If you don't see both of these as described, then you've made a mistake, and you need to follow the zipping instructions above, otherwise marks will be deducted.*

- *You can upload as many attempts as you'd like, but only the final attempt is marked: all other assignments are ignored.*

V. **Marking**

Marks will be awarded according to the following marking scheme.

Students are accorded considerable latitude in this assignment to implement their code in any fashion they deem reasonable, so long as it results in good, clean code. Marks will be deducted if this leniency produces convoluted, buggy code and unreliable execution. This includes code that is excessively sloppy, repetitive, poorly documented, does not use existing code correctly, does not follow standard conventions established in earlier courses, etc. Additionally, you will lose marks for not doing things you should already have learned in your Level I Java course. At this level, you should be writing fairly clean, well-written code, even if this isn't explicitly stated in the marking scheme.

Version Information:

Corrections to this document will be posted as required, so check Brightspace periodically for notification of corrections. Version-specific changes to this document will be listed below.

Item	Marks
File I/O works correctly; you can save/load the employees ArrayList to/from the CurrentEmployees.emp file, stored in the default directory (i.e. the path relative to the folder used for program execution.) This includes correct handling of the FileNotFound exception when the file is missing	**5**
The menu has been altered to include user access to the two new items	**2**
Exception handling has been added to the code so that the user is protected from storing bad data in the employees ArrayList. This includes ensuring that there are no bad dates entered, recovered from mistyped keystrokes (i.e. entering a character where a number was expected.) At the same time, you need to be judicious in your use of exceptions, being careful to avoid using them when a simple if…else… would suffice.	**5**
File labelled and zipped correctly, as per instructions above. The doc folder is supplied in your zip file and contains all classes in your project, hypertext-linked as required	**3**
All classes, methods and constructors have complete JavaDoc comments according to ACDS	**5**
TOTAL :	**20**

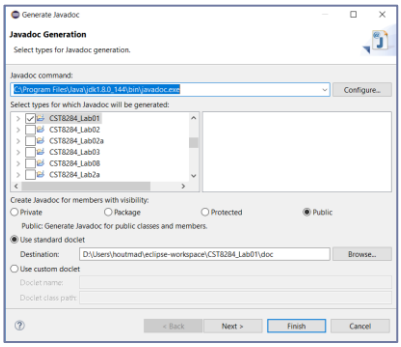
Appendix: Generating JavaDocs

a. To generate JavaDoc, right click the *project* from the Package Explorer. (This is an essential step, and the contents of your Javadoc folder will be incorrect if you do not follow this simple instruction.) From the menu along the top of the Eclipse screen, select Project >> Generate Javadoc…

b. In the dialogue that appears, make sure the assignment is selected from the list of types for which Javadocs will be generated.

c. On occasion, Javadoc needs to be configured. If the Finish button is grayed out, then select the 'Configure…' button, and navigate to the folder in which the JDK is located. The javadoc.exe file is located in the bin directory. Hence your Javadoc path should look like C:\Program Files\Java\ jdk1.8.0_*version*\bin\javadoc.exe

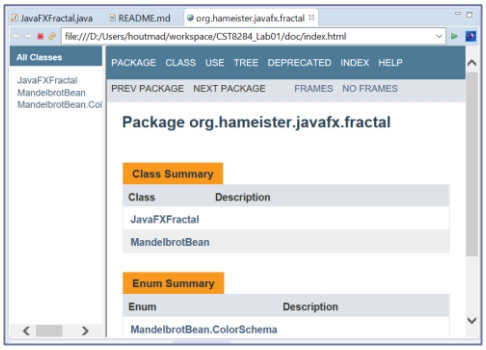
If it doesn't, you may need to navigate to this file and install it manually in Eclipse.



d. Select the Finish button. If you get additional messages during Javadoc setup, select the defaults. (Note that if you have any other tabs open in Eclipse, including malfunctioning code from previous projects, the Javadoc utility may have get confused. Close the tabs for any old files that you've been working on, those which don't have anything to do with your current project. Then run the Javadoc utility again, being careful to select the project file before starting.)

e. Inspect the doc folder which appears in the CST8284_W19_Assignment3 project: this is where your Javadoc files are located.

f. Most of the files in doc are HTML files, which can be opened in Eclipse simply by double-clicking on them. Double click the index.html file (this is the main point of entry for Javadoc files) and a web page, like the one shown below, should appear in the Eclipse code window. Javadoc files are hypertext linked to one another, hence you can click on any link in any Javadoc HTML file to take you to any other file.



If you only see HTML code then you should right click on index.html and select Open With > Web Browser, which should then display the web page rather than the HTML text file it is based on. But if your Eclipse setup is configured correctly, you should see the web page in the Eclipse code window.