**Description**

In this assignment, you will update the Money, Date, Bill, and ArrayList classes from previous homeworks using new ideas that we have discussed in class.  Build a driver that will fully test the functionality of your classes and include the driver with your submission.

1. Fix any privacy (and other) errors that were noted in your comments for the previous iteration of this homework.  (This doesn't include ArrayList; there will be some places where you'll have to have privacy leaks in ArrayList.  You won't be making ArrayList Comparable or Cloneable.)
2. Modify the Money, Date, and Bill class to implement the Comparable interface. Remember that compareTo takes an Object parameter and you should check to make sure that the object class is actually the correct class for the comparison, as appropriate.
3. Modify the Money, Date, and Bill classes to implement the Cloneable interface. Note that Money and Date can simply copy their private instance variables, since they store only primitive and immutable types. However, you will need to override the clone method, to make it public, since it is protected in the Object class. The Bill class will need to do more, since it incorporates the Money and Date classes, which are mutable. Note that it can (and should) use the clone methods of those classes. Be sure to remove any use of the copy constructor for Money, Date, and Bill in the rest of the code (the definition can exist, but don’t use it in other classes; use the clone method instead).
4. Build a class ExpenseAccount that extends (i.e., inherits from) your ArrayList. You should remove the limit on the number of bills that can be placed in an account by making your ArrayList dynamically resize itself.
5. Modify Money, Bill, and Date to implement the Serializable interface.
6. Have your ArrayList class implement the Iterable interface.  Implement the Java 7 specification, not later specifications like Java 8.  That is, see:  [http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html (Links to an external site.)](http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html).  The Java 8 specification includes resources to support functional programming and lambda expressions which is out of the scope of this class.

**Hints, Etc.**

* Remember to follow the Coding Style Guidelines.
* **Make sure you comment all methods and the class with javadoc comments.**  This includes constructors, getters, setters, etc.  If you have files based on code someone else has written (e.g., is based on a skeleton), you have to javadoc comment the methods someone else wrote also.  You cannot receive full-credit for the rubric's commenting criterion if you do not have a **javadoc comment for every method,** no matter how small.
* Your code should not depend on absolute path specifications or any other environment-specific specifications.  The program should **not** ask for user input from the console.
* The description for this assignment is much briefer than in some previous assignments. In particular, not much detail is given regarding what methods to have, etc. **It is purposely open-ended.** By this point, you should be able to design an appropriate interface to your classes as well as create an appropriate driver filled with tests. My advice is not to scrimp on the interface; include all methods one would reasonably expect objects of that class would be able to do.
* An example of questions to ask when defining an interface (for the ExpenseAccount class):  What kind of data and methods should go into an ExpenseAccount class?  What would you use an ExpenseAccount for?  What would you want to inquire of an expense account?  (If you don't know what an expense account is, do some web searches to find out.)  For instance, you might be interested in knowing the total amount of expenses that is outstanding in the ExpenseAccount.  You might be interested in knowing which expense is next due and unpaid and how much it is.  And so on ... these are just two examples of questions you might ask of an expense account.  It depends on the use cases you want the expense account to satisfy.  These kinds of questions can also be asked for Bill, etc.

**What to Submit**

Classes whose names are given in this assignment description should be given that name.  Please **submit your files inside a zip archive called ClassesInterfaces.zip.**

**About This Document**

Original assignment by Rob Nash, Autumn 2014. Minor edits and additions by Johnny Lin, November 2019.

**Rubric**

Interfaces Rubric (2)

| Interfaces Rubric (2) | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning Outcome Money is Comparable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Bill is Comparable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Date is Comparable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Money is Cloneable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Bill is Cloneable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Date is Cloneable |  | 4.0 pts |
| This criterion is linked to a Learning Outcome ExpenseAcount subclass |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Serializable Interface for Money, Bill, & Date |  | 2.0 pts |
| This criterion is linked to a Learning Outcome Clear, Well-Written, and Complete Comments in Code |  | 8.0 pts |
| This criterion is linked to a Learning Outcome Program Compiles and Properly Runs |  | 4.0 pts |
| This criterion is linked to a Learning Outcome ArrayList Implements Iterable |  | 4.0 pts |
| Total Points: 46.0 | | | |

**Submission**

Submitted!

Nov 21 at 11:58pm

[Submission Details](https://canvas.uw.edu/courses/1331881/assignments/4923141/submissions/3729452)

[Download ClassesInterfaces.zip](https://canvas.uw.edu/courses/1331881/assignments/4923141/submissions/3729452?download=60037681)

Grade: 38 (46 pts possible)

Graded Anonymously: no

[View Rubric Evaluation](https://canvas.uw.edu/courses/1331881/assignments/4923141/submissions/3729452#rubric)

**Comments:**

compareTo should take an Object as a parameter and should handle null as well as Objects of incorrect type. Your Bill class throws a NullPointerException if null is passed into compareTo. Bill throws NullPointerException when cloning an unpaid Bill. No iterable implementation. Unfixed error in Money.