Regis University CC&IS
CS310 Data Structures
Programming Assignment 2: Arrays and ArrayLists

Problem Scenario

The real estate office was very impressed with your work from last week. The IT director now needs you to modify the program so it will store data about multiple realtors and properties.

Program Requirements

Last week, you created your **Realtor** and **Property** classes. This week you will be implementing arrays and ArrayLists containing objects from those classes. The two data structures you will implement are:

An ordered ArrayList, ***in ascending order*** by realtor license number, containing **Realtor** objects.
An unordered array, containing **Property** objects.

The program must follow the **CS310 Coding Standards** from Content section 1.9.

Input Data File

This **csv** file input file for this program will be called "assn2input.txt", and will be modified slightly from last week. The file still will contain lines of data about realtors and properties. But the second data item on each line that describes an action to take, will now indicate whether to **ADD** or **DEL** (delete) the specified realtor or property.

If the action is "ADD", then the remaining data fields will be the same as in assignment 1.
But if the action is "DEL", the data line will contain only one more data field:
a license number for realtors, or an MLS number for properties.

The format will be (any of the following data lines, in any order):

```
REALTOR,ADD,licenseNum,firstName,lastName,phoneNumber,commissionPercentage

REALTOR,DEL,licenseNum

PROPERTY,ADD,mlsNum,licenseNum,streetAddr,city,state,zipCode,numBedrooms,
numBathRooms,sold(Y/N),askingPrice

PROPERTY,DEL,mslNum
```

Note that a REALTOR, ADD line must appear before any lines to add Property objects for that Realtor, because a Property can only be added for ***existing*** Realtors.

See the sample input file provided in **Appendix A** below, to see how the data lines will look. You will need to create your own test input data files to test all aspects of your program.

When you created your project last week, NetBeans created the initial source file, which contained the **main** method (**CS310<lastname>.java**). We will refer to this as the "main class". The code to where the program starts execution will be in the **main** method of the **main** class file, **CS310<lastname>.java** file.

Domain Classes

This program will no longer use the **setRealtorAttributes** and **setPropertyAttributes** methods that you used to test your ***setters*** in the previous assignment. Instead, you will convert the methods to be ***constructors*** in the **Realtor** and **Property** classes.

Each of these constructors will have an array of String values as a parameter (containing the data line read from the input file), and will use the values in the array to initialize each attribute value in the new object. Note that you will also need to modify the code in the main class that instantiates your Realtor and Property objects to use this new constructor, instead of using the default empty constructor, and eliminate the calls to the set attributes methods.

After creating the new constructors, you can delete the **setRealtorAttributes** and **setPropertyAttributes** methods from the main class. You can also delete the **displayRealtorAttributes** and **displayPropertyAttributes** that were used to test the ***getters*** last week.

Implementation Classes

You will add separate ***implementation*** classes to implement the array and ArrayList. Your **main** class file will not know what data structure is being used. In fact, this **main** class file will be used with future assignments, and will process the same (or a similar) input file format each time.

The ***implementation*** classes will also be modified each week, to reflect whichever data structure is being used that week. This week you will create the following classes for implementation of the unordered array and ordered ArrayList:

RealtorLogImpl and **PropertyLogImpl**

Note that appending "Impl" to the end of an implementation class name is a standard.
We have also added "Log" for now, to aid in your understanding of what is being implemented (i.e. a log of realtors and a log of properties).

The **RealtorLogImpl** class will be used to create and manage an ArrayList that holds an ***ordered*** list of Realtor objects (ordered by ascending realtor license number). The class will have one private attribute, the ArrayList of Realtor objects.

The class will minimally contain the following methods:

```
public ArrayList<Realtor> getRealtorLog      // return the ArrayList attribute

public void add(Realtor obj)                 // add realtor object to ordered list

public boolean remove (String license)// remove Realtor with specific license from list
                                      // and return true if successful

// test if Realtor with specific license exists in log
public boolean isLicenseUnique(String license)
```

Since the ArrayList of Realtors is an ***ordered*** list, when adding an object to the log, you will need to search for the correct index location to add it to (i.e. the index ***after*** the index containing the largest license that is smaller than the license being inserted). HINT: Use String method **compareTo().**

The **PropertyLogImpl** class will be used to create and manage an ***unordered*** array that holds Property objects.

The maximum number of Property objects that can be stored in the array will be 1000 (which should be defined as a constant within the class).

The class will have two private attributes, an array of Property objects and a count of how many Property objects are stored in the array.

The class will minimally contain the following methods:

```
public Property[] getPropertyArray    // return the array attribute
public int getNumProperties    // return the count attribute

// add Property to log if there is room, and return true if successful
public boolean add(Property obj)

// delete all Property objects from log for realtor with license
//        & return true if any Property objects were deleted
public boolean remove(String license)

// overloaded - remove Property with mlsNum from log
public boolean remove(int mlsNum)

public boolean isMlsUnique(int mlsNum)    // test if Property with mlsNum exists in log

// return count of Property objects in log with specific realtor license
public int numberOfProperties(String license)

public double totalPropertyValue()            // return sum of all asking prices

// overloaded - return sum of asking prices for specific realtor license
public double totalPropertyValue(String license)
```

Since the array of Property objects is an ***unordered*** list, when adding a Property object to the log, you can just add to the end of the log. When deleting a Property object from the log, you can just replace the object being deleted with the last object in the log and decrement the count.

The program will also have a **PrintImpl** class, which will contain a method to generate a report (more below).

You may also add any other additional methods that you think you need to your implementations.

The main class

All implementations will be instantiated as objects in the **main *class***. For example:

```
static RealtorLogImpl realtorLogImpl = new RealtorLogImpl();
```

In this example, the **`realtorLogImpl`** object has a single attribute, which is the ArrayList of realtors.

In addition to the ***implementation*** instantiations, the **main *class*** will contain the following static methods:

- Method to read the data file and process each data line
  - Try to open the input data file and throw/handle an exception if the file cannot be opened.
  - If the file opens, for each line in the data file:
    - Read the data line.
    - Use **split**() to parse the line to extract the individual data fields into an array.
    - Depending on the String values in the array at:
      - index 0 (REALTOR or PROPERTY)
      - index 1 (ADD or DEL)

      Call a method to process the addition or process the deletion, passing in the parsed String array as a parameter.

- Method to process a **Realtor addition** to the list
  - Create a new **Realtor** object using the new constructor and String array parameter values passed into this method.
  - Validate the realtor license number and phone number using the methods you wrote for Assn 1.
    - If either is not valid, display an error message that includes the realtor license number.
    - If the phone number is invalid, include the bad phone number too.

      NOTE: Invalid data will ***not*** prevent the Realtor from being added to the ArrayList.
  - Check to see if Realtor license number is unique using the Realtor **isLicenseUnique**() method.
    - If the Realtor license number is unique,

      Call Realtor **add** to add the Realtor object to the Realtor ArrayList data structure and display a message that includes the realtor license number, stating that the Realtor has been added to the log.
    - Otherwise,

      Realtors with non-unique Realtor license numbers will ***not*** be added to the Realtor ArrayList. So display an **error** message that includes the realtor license number, stating the Realtor will **not** be added to the log.

  NOTE: You cannot use **==** to compare Strings. You must use a String class method.

- Method to process a **Property addition** to the list
  - Create a new **Property** object using the new constructor and the String array parameter values passed into this method.
  - Validate the MLS number, state, and zip code using the methods you wrote for Assn 1.
    - If any of them are invalid, display an error message that includes the MLS number and any additional bad data.
    - NOTE: Invalid data will not prevent the Property from being added to the array.
  - Check to see if the Realtor license number is NOT unique using the Realtor **isLicenseUnique**() method.
    - When NOT unique, the Realtor is in the Realtor list, so a Property can be added for that Realtor.
  - Check if the Property MLS number is unique using the Property **isMlsUnique** () method.
    - When unique, the MLS number is not already in the Property list and the Property can be added.
  - If the Realtor license number is ***not*** unique AND the Property MLS number ***is*** unique,

    Call Property **add** to add the Property object to the Property array data structure and display a message that includes the realtor license number and MLS number, stating that the Property has been added.
    - Otherwise,

      Property objects with unique realtor license numbers or non-unique property MLS numbers will ***not*** be added to the Property array.
      So display an error message stating that the Property will not be added to the log, and explain why (bad Realtor license or bad MLS number).

- Method to process a **Realtor deletion**
    - Check if the Realtor license number is NOT unique using the Realtor **isLicenseUnique**() method.
        - If the Realtor **is** in the Realtor list, it can be deleted.
    - If the Realtor license number **is** in the list:
        - Call Realtor **remove** to find the Realtor object with the correct Realtor license number and remove that Realtor object from the Realtor ArrayList data structure.
            - Display a message confirming which Realtor license number was deleted, and that all the realtor's properties will also be deleted.
        - Call Property **remove** to delete all Property objects associated with the Realtor license number that has been deleted.
    - Otherwise issue an error message that the Realtor license number was not found in the log.

- Method to process a **Property deletion**
    - Check if the Property MLS number is NOT unique, using the Property **isMlsUnique**() method.
        - If the Property **is** in the Property list, it can be deleted.
    - If the Property MLS number **is** in the list:
        - Call Property **remove** to:
            - Find the Property object with the correct Property MLS number
            - Remove that Property object from the Property array data structure.
        - Display a message confirming which Property MLS number was deleted.
    - Otherwise issue an error message that the Property MLS number was not in the log.

See the sample display output in the **Appendix A** below, to see what will be output while reading and the data file and calling methods to process each line.

- Method to **create a report** (see requirements below)

The main class will also (obviously) still contain a **main** ***method***. Remove all the testing code (created for Assn 1) from the **main** method, and instead add code to:
- Call the method to read and process the data file.
- Call the method to create the report.

Output Report

After all input has been read and processed, the real estate office would like you to create a report. You will use a print implementation class, **PrintImpl**, to do this.

The **PrintImpl** class will contain a single method that will create the report. The method will have both implementation objects as its input parameters, and will use those objects with getters to populate the local data fields, within the **PrintImpl** class.

The report will be written to an output file named "assn2report.txt", which will be located in the **output** folder of the project. The **PrintImpl** class should have a constant to hold this filename, as follows:

```
   final String OUTPUT_FILENAME = "output/assn2report.txt";
```

The **PrintImpl** class will have three data fields, one for an ArrayList of Realtor objects, one for the array of Property objects, and one for a count of Property objects in the array. Each of these data fields should be populated using the ***getters*** from the other implementation methods.

To create the report:

For each Realtor in the Realtor ArrayList, the code will examine each Property in the Property array and find Property objects with realtor licenses that match the realtor license of the Realtor object.

HINT: Use nested loops.

So for ***each*** Realtor, the report will include the following:

```
RealtorLicense RealtorLastname, RealtorFirstname

        MLSnumber                  streetAddress  bedrooms/bathrooms  askingPrice  SOLD*
                           city, state, zipCode
              :
              :
        MLSnumber                  streetAddress  bedrooms/bathrooms  askingPrice  SOLD*
                           city, state, zipCode

    Number of Property Listings for Realtor: totalNum
    Total sales value of Property Listings for Realtor:  $ totalValue
```

*only display SOLD if true

At the end of the report, the program will also display the total number of Property Listings for ***all*** Realtors, and the total sales value of all Property Listings for ***all*** Realtors. And after creating the report, the program should tell the user which file the report was stored in.

See the sample output in the **Appendix A** below, to see how the formatting will look.

Additional Requirements

- Create **Javadoc headers**, and generate **Javadoc files** for each of your new ***implementation*** classes and for all new static methods in the main class. You are responsible for completing the comment headers created.
- Your input data file will still be read from the **input** folder in your project.

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:
  `assn2input1.txt`
  `assn2input2.txt`
  (i.e. number each data file after the filename of **`assn2input.txt`**)

  Together, all of your test data files should demonstrate that you have tested every possible execution path within your code, including erroneous data which causes errors or exceptions.

  For example, make sure you have test files with Realtor license numbers that are ***not*** in the correct order, so you can validate that your program is inserting them in order.

  WARNING: You will not be given ANY credit for the input file that is included as an example with these requirements. The test files should be of your own creation.

- You will need to add another folder to your project for this assignment, called "**output**". The report your program creates will be written to the **output** folder.

- Add screen shots of **clean compile** of your classes to the documentation folder.
  - Remember, if you have compile errors, a red error symbol is placed on the file and folder name tabs. Be sure to clear all compile errors before capturing the screen shot.

  WARNING: Submittals without the clean compile screenshots will **not** be accepted.
  (This means that programs that do not compile will **not** be accepted)

Program Submission

This programming assignment is due by midnight of the date listed on the **Course Assignments by Week** page of the Content.

- Export your project from NetBeans (same as Assn 1):
  - Highlight the project name.
  - Click on **File** from the top menu, and select **Export Project**.
  - Select **To ZIP**
  - Name your export file in the following format:
    `CS310<lastname>Assn<x>.zip`

    For example:
    `CS310SmithAssn2.zip`

  NOTE: Save this zip file to some other directory, not your project directory.

- Submit your **.zip** file to the **Prog Assn 2** Submission Folder (located under **Assignments** tab in online course).

  Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

This program will be graded using the **rubric** that is linked under **Student Resources** page.

WARNING:
Programs submitted more than 5 days past the due date will ***not*** *be accepted, and will receive a grade of 0.*

See Appendices on next pages for sample inputs and outputs, and a sample code outline.

Appendix A – Sample Input and Output

Sample Input File

```
REALTOR,ADD,AA1111111,Anna,Astrid,111-111-1111,0.011
PROPERTY,ADD,1111111,AA1111111,111 Main St,Denver,CO,80221,2,1,Y,111111
PROPERTY,ADD,11111222,AA1111111,112 Main St,Denver,CO,80221,1,1,N,111111
REALTOR,ADD,BB2222222,Bob,Benson,222-222-22222,0.012
PROPERTY,ADD,2222211,BB2222222,222 Flower Ln,Somewhere,WX,802211,3,1.5,N,222222
PROPERTY,ADD,2222222,BB2222222,1234 First St,Elsewhere,CO,80221,4,2,Y,222222
PROPERTY,DEL,2222221
REALTOR,ADD,CC333333X,Carl,Carlson,333-333-3333,0.013
PROPERTY,ADD,3333333, CC333333X,222 Flower Ln,Somewhere,WX,80221,4,3,Y,333333
REALTOR,DEL,CC333333X
```

Sample Display Output

```
Reading data from file input/assn2Input.txt
  ADDED: Realtor with license AA1111111
  ADDED: Property with MLS number 1111111 listed by realtor AA1111111
    ERROR: Property with MLS number 11111222 has an invalid MLS number.
  ADDED: Property with MLS number 11111222 listed by realtor AA1111111, regardless of data errors.
    ERROR: Realtor with license BB2222222 has invalid phone number 222-222-22222
  ADDED: Realtor with license BB2222222, regardless of data errors.
    ERROR: Property with MLS number 2222211 has invalid state abbreviation WX
    ERROR: Property with MLS number 2222211 has invalid zip code 802211
  ADDED: Property with MLS number 2222211 listed by realtor BB2222222, regardless of data errors.
  ADDED: Property with MLS number 2222222 listed by realtor BB2222222
  DEL ERROR:  Property with MLS number 2222221 is not in the property log, so it cannot be deleted.
    ERROR: Realtor with license CC333333X has invalid license number.
  ADDED: Realtor with license CC333333X, regardless of data errors.
    ERROR: Property with MLS number 3333333 has invalid state abbreviation WX
  ADD ERROR: Property with MLS number 3333333 has Realtor with license  CC333333X,
             but there is no such Realtor license in the realtor log.
             So property will NOT be added to property log.
  DELETED: Realtor with license CC333333X has been removed from the realtor log
            All realtor's properties will also be removed from the property log
Done reading file. 10 lines read

Creating report...
Report is located in file: output/assn2report.txt
```

Sample Report Output

```
AA1111111  Astrid, Anna

        1111111         111 Main St    2/1.0  $  111111.00  SOLD
                  Denver, CO 80221

       11111222         112 Main St    1/1.0  $  111111.00
                  Denver, CO 80221

    Number of Property Listings for Realtor: 2
    Total sales value of Property Listings for Realtor: $ 222222.00

BB2222222  Benson, Bob

        2222211       222 Flower Ln    3/1.5  $  222222.00
              Somewhere, WX 802211

        2222222       1234 First St    4/2.0  $  222222.00  SOLD
                Elsewhere, CO 80221

    Number of Property Listings for Realtor: 2
    Total sales value of Property Listings for Realtor: $ 444444.00

Total Number of Property Listings for ALL Realtors = 4
Total sales value of Property Listings for ALL Realtors = $ 666666.00
```

Appendix B – Partial Sample Code outline

Sample main class

```
public class CS310Lastname {
    static RealtorLogImpl realtorLogImpl = new RealtorLogImpl ();
    static PropertyLogImpl propertyLogImpl = new PropertyLogImpl ();
    static PrintImpl printImpl = new PrintImpl();

    public static void main(String[] args) {
        processFile();
        createReport();
    }

    public static void processRealtorAddition (String [] inputLineValues) {
      :
    }

    public static void processPropertyAddition (String []inputLineValues) {
      :
    }

    public static void processRealtorDeletion(String [] inputLineValues) {
      :
    }

    public static void processPropertyDeletion(String [] inputLineValues) {
      :
    }

    private static void processFile() {
       // Try to open file and throw exception if file not found
       // Loop:
       //   Read and parse data line
       //   Call processRealtorAddition, processPropertyAddition, processRealtorDeletion, or processPropertyDeletion
    }

    public static void createReport() {
       // Call method from PrintImpl class to print the report
    }

    // Any other static methods that you would like to define
      :

}
```