Regis University CC&IS
CS310 Data Structures
Programming Assignment 1: Create class domains in NetBeans and test them

Problem Scenario

A Colorado real estate office is in need of a Java programmer to assist them with some of their needs. The office needs to be able to track information about their realtors and the properties listed for sale. All properties are located in Colorado and Wyoming.

For each **Realtor**, the real estate office needs to know:

- Realtor's license number (Unique String – two letters followed by seven digits)
- Realtor's first name (String)
- Realtor's last name (String)
- Realtor's phone number (String formatted as "###-###-####")
- Realtor's commission (double representing a percentage)

The real estate office has had issues with incorrect realtor license numbers and phone numbers, so they are requesting a way to be able to determine if the they are correct.

For each **Property**, the real estate office wants to track:

- MLS number (Unique 7-digit integer that does NOT begin with a 0)
- Realtor's license number (to match a Realtor)
- Property street address (String)
- Property city (String)
- Property state (2-char uppercased String containing CO or WY)
- Property zip code (5-digit integer)
- Number of bedrooms (integer)
- Number of bathrooms (double)
- Sold (boolean)
- Asking price (double)

The real estate office has had issues with incorrect MLS numbers, state abbreviations, and zip codes, so they are requesting a way to be able to determine if the they are correct.

Program Requirements

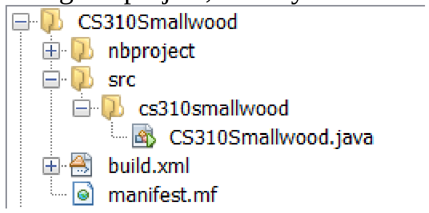
The assignment this week is to create and test some of the classes for the problem scenario above. These classes will be used to implement data structures in later assignments.

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

To set up a project directory for your program:

- Create a new **project** in NetBeans, and name it **CS310 + last name** (e.g. `CS310Smith`).
- Use the ***default*** Main Class name provided by NetBeans (e.g. `cs310smith.CS310Smith`). This will cause NetBeans to create a **package** for your source code files.
  - NetBeans will create the default folders for the project.
  - NetBeans will create the **main** method within the main class (e.g. `CS310Smith.java`)

After creating the project, when you view the **Files** tab, the project structure will look like this:



You will create two additional classes in the same **CS310<lastname>** folder as the **CS310<lastname>.java** file (click on the **CS310<lastname>** folder before selecting **New File**). The additional classes are:

- a **Realtor** class
- a **Property** class

These classes will be used for creating and manipulating the **Realtor** and **Property** objects, and will ***not*** contain a **main** method.

After you create the **Realtor** and **Property** classes, use the tips provided in the NetBeans reader to generate the following methods for both classes:

constructors
getters/setters
equals() – must compare ALL attribute values
toString() – must include ALL attribute values

The **Realtor** and **Property** classes should include ***two*** constructors each:

an empty constructor
a constructor that accepts all of the attributes of the class as parameters

While NetBeans can help you create the basic methods, you will also need to add additional data error checking methods, as follows:

Error checking methods for the **Realtor** class:

- Check the license contains 2 letters followed by 7 digits within the String.
- Check the phone number is 12 chars long and contains dashes and digits in the correct places

Error checking methods for the **Property** class:

- Check the MLS number is 7 digits long
- Check the state is either “CO” or “WY”
- Check the zip code is a Colorado zip code (5 digits long, starting with 80, 81, 82, or 83).

All of these error checking methods will simply check the proper attribute and return a **true** or **false**.

Input Data File

You will use a simple **csv** file input file for this program (see the online **Content** for information on how to read a **csv** file). The file will contain four lines of data about Realtors and Property

This week’s input file, **assn1input.txt**, will contain ***only 4 data lines***, in the following format:

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

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

The data will be two lines of **Realtor** data, followed by two lines of **Property** data.

Sample Input File Lines

```
REALTOR,ADD,AB1234567,Jose,Ferguson,303-333-3333,0.018

REALTOR,ADD,DK4567890,Antonia,Swarez,720-444-4444,0.020

PROPERTY,ADD,7654321,AB1234567,567 Main St,Denver,CO,80221,3,1.5,N,222333

PROPERTY,ADD,9876543,DK4567890,111 Hover St,Longmont,CO,80501,4,2,Y,321321
```

This file will be placed in an **input** directory within your project (see Deliverables section below for how to create new directories within your project). Within your code, define a constant to hold this filename, as follows:

```
final String INPUT_FILENAME = "input/assn1input.txt";
```

Testing

This week, the **main** method in the **`CS310Lastname`** class will be used to test your **Realtor** and **Property** classes. The **main** method will run 3 sets of tests.

NOTE: Before each test, display a message stating which test is being run.
Also display a blank line between each of the tests.

Test Set 1

The first set of tests will test the **constructors *with parameters for each attribute*** and the **toString**() method for each class.

Test 1a:
Instantiate a **Realtor** object by hardcoding argument values of your choice in the call to the constructor. Then use the **toString**() instance method to print the attribute values to the console.

Test 1b:
Instantiate a **Property** object by hardcoding argument values of your choice in the call to the constructor. Then use the **toString**() instance method to print the attribute values to the console.

Example:

```
MyClass myObjectName = new MyClass("abc", 123, false);
System.out.println( myObjectName.toString() );
```

Test Set 2

The second set of tests will test the ***empty*** **constructors**, **setters** and **getters**, and will also test reading data from the input data file.

Before running the tests, try to open the input data file.

- Throw an exception of the data file cannot be found.
- The exception handler should display an error message using **`System.err`** instead of **`System.out`** and include the name of the file that could not be found in the message.
- Exit the program with error code 1.

If the file opens successfully, run the rest of the tests that follow.

Test 2a:
Instantiate a **Realtor** object, using the ***empty*** constructor.

Read the ***first*** line of data from the input data file.

Use **split()** to parse the line read into an array of String values.

The first item on the data line should be the String "REALTOR". If it is:

Call a static **setRealtorAttributes**() method. This method will:

- Pass in the **Realtor** object and array of String values as parameters
- Use the setters to set each attribute value
- Return the **Realtor** object with all attributes set.

Call the method to validate the realtor's license number and phone number

- If the either of these items are invalid, display an error message explaining what is invalid.

Call a static **displayRealtorAttributes**() method. This method will:

- Pass in the **Realtor** object as a parameter.
- Use the getters to display each attribute value, one per line.

Otherwise, display an error message that the data is not REALTOR data.

Test 2b:
Instantiate another **Realtor** object, using the ***empty*** constructor.

Read the ***second*** line of data from the input data file.

Use **split()** to parse the line read into an array of String values.

The first item on the data line should be the String "REALTOR".

Check this and repeat all the steps from test **2a**.

Test 2c:
Instantiate a **Property** object, using the ***empty*** constructor.

Read the ***third*** line of data from the input data file.

Use **split()** to parse the line read into an array of String values.

The first item on the data line should be the String "PROPERTY". If it is:

Call a static **setPropertyAttributes**() method. This method will:

- Pass in the **Property** object and array of String values as parameters
- Use the setters to set each attribute value
- Return the **Property** object with all attributes set.

Call the methods to validate the MLS number, state, and zip code

- If the any of these items are invalid, display an error message explaining what is invalid.

Call a static **displayPropertyAttributes**() method. This method will:

- Pass in the **Property** object as a parameter.
- Use the getters to display each attribute value, one per line.

Otherwise, display an error message that the data is not PROPERTY data.

Test 2d:
Instantiate another **Property** object, using the ***empty*** constructor.

Read the ***fourth*** line of data from the input data file.

Use **split()** to parse the line read into an array of String values.

The first item on the data line should be the String "PROPERTY".

Check this and repeat all the steps from test **2c**.

Close the input data file when you have completed the tests in Test Set 2, before running Test Set 3.

Test Set 3
The third set of tests will test the **equals**() methods for both classes.

Test 3a:
Use the **equals**() method to compare the two **Realtor** objects from **Test Set 2**, and display a message stating whether they are equal or not.

Test 3b:
Use the **equals**() method to compare the two **Property** objects from **Test Set 2**, and display a message stating whether they are equal or not.

Notes on Validating your Code

In order to validate your code, you will need to run your test code multiple times, each time using different sets of data. For example:

Run 1: No test data input file in **input** folder.

Run 2: In the file data
For the Realtor objects, one realtor license is invalid and one phone numbers is invalid.
For the Property objects, one MLS number is invalid, one zip code is invalid,
and at least one property is not yet sold.
Neither the Realtor or Property objects are equal (at least one attribute value differs).

Run 3: All data is valid, the Properties are sold, and the Realtor and Property objects are equal (all attribute values are the same).

You must supply enough test files to test all possible ways the tests might fail.

Sample Output Display
```
Running Test 1a:
Realtor{licenseNum=AA1111111, firstName=John, lastName=Smith, phoneNum=303-333-
3333, commissionRate=0.18}

Running Test 1b:
Listing{mlsNum=1234567, licenseNum=AA1111111, streetAddress=1234 Main St,
city=Denver, state=CO, zipCode=80221, numBedrms=3, numBaths=1.5, sold=false,
askingPrice=234500.0}

Running Test 2a:
ERROR:  Invalid phone number format: 303333-3333
AB1234567
Jose
Ferguson
303333-3333
0.018
```

```
Running Test 2b:
ERROR:  Invalid license format: D4567890
D4567890
Antonia
Swarez
720-444-4444
0.02

Running Test 2c:
ERROR:  Invalid state: CY
7654321
AB1234567
567 Main St
Denver
CY
80221
3
1.5
false
222333.00

Running Test 2d:
ERROR:  Invalid MLS number: 987654
ERROR:  Invalid zip code: 84501
987654
DK4567890
1111 Hover St
Longmont
CO
84501
4
2.0
true
321321.00

Running Test 3a:
Realtor objects are NOT equal

Running Test 3b:
Property objects NOT are equal
```

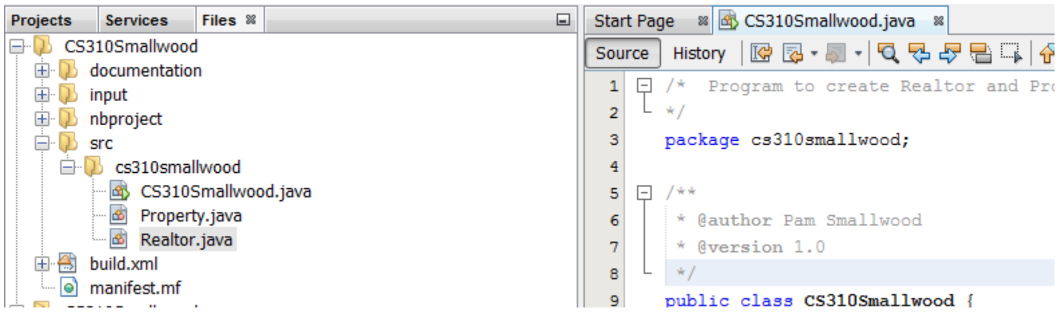
Additional Requirements

- Use NetBeans to build comment headers. You are responsible for completing the comment headers created. Refer to the NetBeans reader on how to use NetBeans to insert your comment headers.

- Add an **input** folder. To create a new folder in NetBeans:
  - Right click on the project name (top level folder).
  - On the next dialog box, select **New**.
  - Another dialog box will pop up. Look for "Folder" (the order of the list varies based on what the last actions you have done). Select **Folder**.
  - When prompted, name the new folder "**input**".

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:
  `assn1input1.txt`
  `assn1input2.txt` (i.e. number each data file after the filename of **`assn1input.txt`**)
  Together, all of your test data files should demonstrate that you have tested all possible code paths.

- Use the same method as above to add a **documentation** folder.
    - Add screen shots of a clean compile of each of your classes to the **documentation** folder. Example:



```
CS310Smallwood
  documentation
  input
  nbproject
  src
    cs310smallwood
      CS310Smallwood.java
      Property.java
      Realtor.java
  build.xml
  manifest.mf
```

```
/*  Program to create Realtor and Pr
*/
package cs310smallwood;

/**
 * @author Pam Smallwood
 * @version 1.0
 */
public class CS310Smallwood {
```

    - 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.
    - You can paste your image file into the documentation folder, as follows:
        - Right click on the **documentation** folder name, and select Paste.

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.

You will submit a zip file containing your project files. To submit your project:

- First export your project from NetBeans. To do this:
    - 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:
      `CS310SmithAssn1.zip`

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

- Then submit your **.zip** file to the **Prog Assn 1** Submission Folder (located under **Assignments** tab in online course).

  Warning: 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.*