THIS IS JUST A FOR EXTRA PRACTICE. DO NOT SUBMIT TO THE SUBMIT SERVER

Specifications

For this assignment you will implement methods for the classes **MultipleChoice, ArrayUtilities**, **House** and **Neighborhood**. You will find the classes in the **sysImplementation** package. A description of each method is provided below. A driver (that you can ignore if you know what to implement) and associated output is provided at the end. This driver is not part of the code distribution. Regarding the code you need to implement:

- You don't need to add comments to your code, but you must have good variable names, indentation and you should avoid code duplication.
- At this point you may want to look at the classes you will find in **sysImplementation**. We have provided a shell for each method you need to implement. You can also see the instance variables associated with each class.
- During the implementation of the above classes, you can add instance variables, constants (static final) and private methods.
- You can assume parameters are valid, unless we indicate otherwise (e.g., if the parameter is null throw …).
- You can create a StringBuffer out of another (e.g., new StringBuffer(anotherStringBuffer)) or by using a string (e.g., new StringBuffer(aString)).
- You must provide an implementation for every method you need to implement, otherwise your code will not work in the submit server. If you don't know what to do for a method, leave the body empty (void method) and for a method returning a value, return any value that makes the code compile.

MultipleChoice Class Specification

This class provides multiple choice questions, where each method represents a question. To answer a question, replace the statement

throw new UnsupportedOperationException("Not Implemented");

with a return statement that returns a character. The following is an example of the kind of question we will provide, and the answer you need to provide:

/* What we will provide */

```
public static char question1() {
        /* Question #1

           This course is:

           a. cmsc106
           b. cmsc131
           c. cmsc200
           d. None of the above


        */
        throw new UnsupportedOperationException("Not Implemented");
}
```

/* You will replace the throw statement with a return and a character */

```
public static char question1() {
        /* Question #1

           This course is:

           a. cmsc106
           b. cmsc131
           c. cmsc200
           d. None of the above


        */
        return 'b';
}
```

ArrayUtilities Class Specification

This class defines the methods:

1. **getFirstAndLastRowCopie**s - The method's prototype is provided below. The method returns a two-dimensional array with two rows, where the first row and second row, are copies of the first and last row of the **data** array, respectively. If the **data** array has less than 2 rows, the method will return null. The **data** array is a ragged array (where the size of a row will be at least 1) and **data** will never be null. This method will not throw any exceptions.

```
public static char[][] getFirstAndLastRowCopies(char[][] data)
```

2. **getIndexFirstRowAllSameCharacter** - The method's prototype is provided below. The method returns the index of the first row of the **data** array where all the characters in the row are the same, and -1 if there is no row where all the characters are the same. The **data** array is a ragged array (where the size of a row will be at least 1) and **data** will never be null. A row with only one character is considered to be a row where all the characters are the same. This method will not throw any exceptions.

```
public static int getIndexFirstRowAllSameCharacter(char[][] data)
```

House Class Specification

The **House** class represents a house. A house has an address (**address** instance variable), a year it was built (**built** instance variable), and the names of residents of the house (**residents** instance variable). The declaration of each variable follows.

```
private String address;
private int built;
private StringBuffer residents;
```

The class methods are:

1. **Constructor** - Takes a string (representing the address) and an integer (representing the year the house was built) as parameters, and initializes the corresponding instance variables, accordingly. It creates a StringBuffer object and assigns it to the **residents** instance variable. You can assume the parameters are valid (e.g., address is not null)
2. **Copy Constructor** - Creates a deep copy.
3. **addResident** - Appends the provided string parameter to the **residents** Stringbuffer if the string parameter is not null. Do not add any delimeter (e.g., a comma) as part of the append process. The method always returns a reference to the current object and no exception will be thrown by this method.
4. **getAddress** - get method for **address**.
5. **getBuilt** - get method for **built**.
6. **toString()** - We have provided this method; do not modify it, otherwise you might not pass release / secret tests.

Neighborhood Class Specification

The **Neighborhood** class represents a neighborhood. A neighborhood has a name (**name** instance variable), an array of references to **House** objects (**houses** instance variable), and the current number of **House** objects (**currHouses** instance variable). Keep in mind that the current number of houses is not necessarily the same as the length of **houses** array. The declaration of each variable follows.

```
private String name;
private House[] houses;
private int currHouses;
```

The class methods are:

1. **Constructor** - Takes a string (representing the neighborhood's name) and an integer (representing the maximum number of houses we can have) as parameters. It creates an array of **House** references with a size corresponding to the integer parameter. It initializes the current number of houses (**currHouses** instance variable) to 0. Remember that arrays of references are initialized to null by default (you don't need to perform such initialization).
2. **addHouse** - Takes as parameters a string (**address**, representing a house address), an integer (**built**, representing when the house was built), and an integer (**whereIndex**, representing the index value of the array entry that will be used to add the new

House object). The method will create a **House** object and assign it to the specified array entry, if the address parameter is not null, if the **whereIndex** parameter is a valid index (one between 0 (inclusive) and the length of the array – 1 (inclusive)), and if the array entry associated with **whereIndex** is currently set to null (i.e., there is no **House** object associated with that entry). If any of these conditions are not met, the method will throw an IllegalArgumentException (any message is fine). If all conditions are met, the new **House** object will be assigned to the array entry and the number of houses will be increased. The method returns a reference to the current object. Notice that **whereIndex** is an index (not a position in the array), therefore, a **whereIndex** value of 2 means the **House** object will be assigned to the third array entry.

3. **findHouse** - Returns true if there is a house with the specified **address** parameter and false otherwise.
4. **getHouses -** The **addHouse** method may add houses at different positions of the array, and not necessarily from the beginning of the array, in a contiguous fashion. For example, you can have a house in the first array entry, no house in the second array entry (entry set to null) and a house in the next entry. The **getHouses** method returns a new array with copies (deep copies) of the **House** objects present in the **Neighborhood** object. **House** object references will appear contiguously in the returned array and the size of the returned array corresponds to the **currHouses** value.
5. **atLeastTwoHousesBuiltSameYear** - This method will return true if there are at least two houses that were built in the same year and false otherwise. Feel free to use the **getHouses()** method during the implementation of this method (although you don't need to).
6. **toString()** - We have provided this method; do not modify it, otherwise you might not pass release / secret tests

Driver / Expected Output (Feel free to ignore)

Driver

```
public static void main(String[] args) {
   String answer = "";

   House house = new House("CollegePark", 1983);
   house.addResident("Kelly").addResident("George");
   answer += house + "\n";

   House copy = new House(house);
   copy.addResident("Sammy");
   answer += copy + "\n";
   answer += "========================\n\n";

   Neighborhood neighborhood = new Neighborhood("GardensView", 5);
   answer += neighborhood;
   answer += "\n*Adding houses*\n";
   neighborhood.addHouse("Elm St 1", 1986, 1);
   neighborhood.addHouse("Oak St 1", 1971, 3);
   answer += neighborhood;
   answer += "\nFound House: " + neighborhood.findHouse("Elm St 1");

   House[] houses = neighborhood.getHouses();
   answer += "\nHow many: " + houses.length + "\n";
   answer += houses[0] + "\n";
   answer += houses[1] + "\n";
   answer += "========================\n\n";

   char[][] data = { {'a', 'b', 'c', 'd'},
                     {'e', 'e', 'e'},
                      {'h'},
                     {'e', 'e', 'e'},
                     {'i', 'j', 'k', 'l', 'm'} };

   char[][] result = ArrayUtilities.getFirstAndLastRowCopies(data);
   answer += "GettingFirstAndLastRowCopies:\n";
   answer += Arrays.toString(result[0]) + "\n";
   answer += Arrays.toString(result[1]) + "\n";

   answer += "\nGettingIndexFirstRowAllSameCharacter:\n";
   answer += "Index: " +
             ArrayUtilities.getIndexFirstRowAllSameCharacter(data);
   System.out.println(answer);
}
```

Output

```
House [address=CollegePark,
built=1983, residents=KellyGeorge]
House [address=CollegePark,
built=1983,
residents=KellyGeorgeSammy]
========================

Name: GardensView
maxHouses: 5
currHouses: 0
Houses Info:
Empty(No House)
Empty(No House)
Empty(No House)
Empty(No House)
Empty(No House)

*Adding houses*
Name: GardensView
maxHouses: 5
currHouses: 2
Houses Info:
Empty(No House)
House [address=Elm St 1,
built=1986, residents=]
Empty(No House)
House [address=Oak St 1,
built=1971, residents=]
Empty(No House)

Found House: true
How many: 2
House [address=Elm St 1,
built=1986, residents=]
House [address=Oak St 1,
built=1971, residents=]
========================

GettingFirstAndLastRowCopies:
[a, b, c, d]
[i, j, k, l, m]

GettingIndexFirstRowAllSameCharacte
r:
Index: 1
```