Name _____Period_____

1. Consider the following instance variable and method. Method wordsWithCommas is intended to return a string containing all the words in listOfWords separated by commas and enclosed in braces. For example, if `listOfWords` contains `["one", "two", "three"]`, the string returned by the call `wordsWithCommas()` should be "`{one, two, three}`".

```
private List<String> listOfWords;

public String wordsWithCommas() {
     String result = "{";

     int sizeOfList = listOfWords.size();

     for (int k = 0; k <= sizeOfList; k++)   {
          /* Modify the code */
          if(k != sizeOfList-1)
                result = result + ", ";
     }
    result = result + "}";
    return result;
 }
```

Need to get the values from the list

(a) Identify the error in the code above with a circle. Write the corrected code below.

Actually, two lines need to be modified. We also need to assume that listOfWords is initialized somewhere else in the code

(b) Write one line of code that could replace `/* Modify the code */` to ensure the method works as intended.

```
public static String wordsWithCommas() {
     String result = "{";

     int sizeOfList = listOfWords.size();

     for (int k = 0; k < sizeOfList; k++)   {
     //for (int k = 0; k <= sizeOfList; k++)   {    //out of bounds as written
       /* Modify the code */
       result = result + listOfWords.get(k);//need to get the value
         if(k != sizeOfList-1)
             result = result + ", ";

     }//end for
     result = result + "}";

     return result;
  }//end method
```

/4

2. Consider the following code segment.

```
/** @param wordList an ArrayList of String objects
 *  @param word the word to be removed
 *  Postcondition:  All occurrences of word have been removed from
 *  wordList.
 */
public static ArrayList<String> removeWord(ArrayList<String> wordList,
String word){
     for(int i = 0; i < wordList.size();i++){
          if(wordList.get(i).equals(word))
               wordList.remove(i);
     }
return wordList;
}
```

(a) Consider a `wordList` which contains the following values,

`{bat  dog  cat  sat  rat  rat  rat}`

What is returned when the following method is called?

`removeWord(wordList, "rat")`

bat
dog
cat
sat
rat

an instance of rat is still in the list. How could you modify the code above to fix this?

/1

3. A two-dimensional array of integers in which most elements are zero is called a sparse array. Because most elements have a value of zero, memory can be saved by storing only the non-zero values along with their row and column indexes. The following complete `SparseArrayEntry` class is used to represent non-zero elements in a sparse array. A `SparseArrayEntry` object cannot be modified after it has been constructed.

```
public class SparseArrayEntry {
/** The row index and column index for this entry in the sparse array */
     private int row;
     private int col;

/** The value of this entry in the sparse array */
     private int value;

/** Constructs a SparseArrayEntry object that represents a sparse
*   array element with row index r and column index c, containing
*   value v.
*/
public SparseArrayEntry(int r, int c, int v)   {
     row = r;
     col = c;
     value = v;   }

/** Returns the row index of this sparse array element. */
     public int getRow()   {
          return row;
     }

/** Returns the column index of this sparse array element. */
     public int getCol()   {
          return col;
     }

/** Returns the value of this sparse array element. */
     public int getValue()   {
          return value;
      }
}
```

The `SparseArray` class represents a sparse array. It contains a list of `SparseArrayEntry` objects, each of which represents one of the non-zero elements in the array. The entries representing the non-zero elements are stored in the list in no particular order. Each non-zero element is represented by exactly one entry in the list

```
public class SparseArray {
/** The number of rows and columns in the sparse array. */
     private int numRows;
     private int numCols;
/** The list of entries representing the non-zero elements of the
 *  sparse array. Entries are stored in the list in no particular order.
 *  Each non-zero element is represented by exactly one entry in
 *  the list.
*/
     private ArrayList<SparseArrayEntry> entries;
/** Constructs an empty SparseArray. */
     public SparseArray()   {
          entries = new ArrayList<SparseArrayEntry>();
     }

/** Returns the number of rows in the sparse array. */
     public int getNumRows()   {
          return numRows;
     }

/** Returns the number of columns in the sparse array. */
     public int getNumCols()   {
          return numCols;
     }

/** Returns the value of the element at row index row and column
 *  index col in the sparse array.
 *  Precondition:  0 ≤ row < getNumRows()
 *  0 ≤ col < getNumCols()
*/
     public int getValueAt(int row, int col)   {
          /* implementation not shown */
     }

/** Returns the col in which the first instance of an element is found
 *  Precondition:  0 ≤ col < getNumCols()
 *  0 ≤ col < getNumCols()
*/
     public int findCol(int value)   {
          /* to be implemented in part (a) */
     }
/** Removes the row from the sparse array.
 *  Precondition:  0 ≤ row < getNumRows()
 */
     public void removeRow(int row)   {
          /* to be implemented in part (b) */
     }
// There may be instance variables, constructors, and methods that are not
shown.
}
```

(a) The following table shows an example of a two-dimensional sparse array. Empty cells in the table indicate zero values

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | | 5 | | | 4 |
| 2 | 1 | | | | |
| 3 | | -9 | | | |
| 4 | | | | | |
| 5 | | | | | |

The sample array can be represented by a `SparseArray` object, `sparse`, with the following instance variable values. The items in entries are in no particular order; one possible ordering is shown below.

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

Write the `SparseArray` method `findCol`. The method returns the col in which the first instance of the value is found. If there are no cols that contain the specified value, 0 is returned.

In the example above, the call `sparse.findCol(-9)` would return 1, and `sparse.findCol(3)` would return 0.

Complete method `findCol` below.

```
/** Returns the col in which the first instance of an element is found
 *  Precondition:  0 ≤ col < getNumCols()
 *                 0 ≤ row < getNumRows()
 */
public int findCol(int value)
```

```
for(int e = 0; e < entries.size();e++){
    if(entries.get(e).getValue() == value){
        return entries.get(e).getCol();
    }
}
return 0;
```

/3

(b) Write the `SparseArray` method `removeRow`. After removing a specified row from a sparse array:

- All entries in the list entries with row indexes matching row are removed from the list.
- All entries in the list entries with row indexes greater than row are replaced by entries with row indexes that are decremented by one (moved one row to the up).
- The number of rows in the sparse array is adjusted to reflect the row removed.

The sample object sparse from the beginning of the question is repeated for your convenience.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | | | 5 | | 4 |
| 2 | 1 | | | | |
| 3 | | | -9 | | |
| 4 | | | | | |
| 5 | | | | | |

The outlined entries below correspond to the shaded column above.

```
entries     row: 2      row: 1      row: 1      row: 3
            col: 0      col: 2      col: 4      col: 2
            value: 1    value 5     value: 4    value: -9
```

When sparse has the state shown above, the call `sparse.removeRow(1)` could result in sparse having the following values in its instance variables (since entries is in no particular order, it would be equally valid to reverse the order of its two items). The shaded areas below show the changes.

```
numRows: 5
numCols: 5

entries:
```

| | |
|---|---|
| `row: 1`<br>`col: 0`<br>`value: 1` | `row: 2`<br>`col: 2`<br>`value: -9` |

```
numRows: 4
numCols: 5
```

Class information repeated from the beginning of the question

```
public class SparseArrayEntry

public SparseArrayEntry(int r, int c, int v)
public int getRow()
public int getCol()
public int getValue()
```

```
public class SparseArray

private int numRows
private int numCols
private List<SparseArrayEntry> entries
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int col)
public void removeRow(int col)
public int findCol(int value)

Complete method removeRow below.

/** Removes the row from the sparse array.
 *  Precondition:  0 ≤ row < getNumRow()
 */

public void removeRow(int row)
```

```
//First lets remove the entries that correspond to the current row
        for(int e = 0; e < entries.size(); e++){
          if(entries.get(e).getRow() == row){
              entries.remove(e);
              e--;
          }
        }

        //now we will update the entries with a row greater than the current row
        for(int e = 0; e < entries.size(); e++){
          if(entries.get(e).getRow() > row){
             entries.set(e, new SparseArrayEntry(entries.get(e).getRow()-1, entries.get
(e).getCol(), entries.get(e).getValue()));
          }
        }
```

/5