**Question 1**

**25 minutes to complete, 5 minutes to upload answer.**

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 List<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)
   { /* to be implemented in part (a) */ }

   /** Returns an integer array containing each value from each row sequentially (including 0 values),
    *    starting at the first value of the top row, and ending at the last value of the bottom row.
    */
   public int[] serialize()
   { /* to be implemented in part (b) */ }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

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

| | | | | |
|---|---|---|---|---|
| | 5 | | | 4 |
| 1 | | | | |
| | -9 | | | |
| | | | | |
| | | | | |

(a) Write the `SparseArray` method `getValueAt`. The method returns the value of the sparse array element at a given row and column in the sparse array. If the list `entries` contains an entry with the specified row and column, the value associated with the entry is returned. If there is no entry in `entries` corresponding to the specified row and column, 0 is returned.

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

```
numRows: 6
numCols: 5
```

| | | | |
|---|---|---|---|
| row: 1<br>col: 4<br>value: 4 | row: 2<br>col: 0<br>value: 1 | row: 3<br>col: 1<br>value: -9 | row: 1<br>col: 1<br>value: 5 |

(b) Write the `SparseArray` method `serialize`. The method returns a one-dimensional integer array containing all the values (including 0 values) from the array, starting at the first value of the top row, and ending at the last value of the bottom row.

Complete method `getValueAt` below.

```
/** 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)

  {
```

```
/**  Returns an integer array containing each value from each row sequentially (including 0 values),
 *    starting at the first value of the top row, and ending at the last value of the bottom row.
 */

public int[] serialize()

{
```

**Question 2**

**15 minutes to complete, 5 minutes to upload answer.**

This question involves analyzing and modifying a string. The following Phrase class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the Phrase class.

```
public class Phrase
{
      private String currentPhrase;

      /** Constructs a new Phrase object. */
      public Phrase(String p)
      { currentPhrase = p; }

      /** Returns an integer array containing each index of occurrence of str in the current phrase.
      *    returns null if there is no occurrence of str.
      *    Precondition: str.length() > 0 and n > 0
      *    Postcondition: the current phrase is not modified.
      */
      public int[] findAllOccurrences(String str)
      { /* implementation not shown */ }

      /** Modifies the current phrase by replacing each occurrence of str with repl.
      *    If there is no occurrence of str, the current phrase is unchanged.
      *    Precondition: str.length() > 0 */
      public void replaceAllOccurrences(String str, String repl)
      { /* to be implemented */ }

      /** Returns a string containing the current phrase. */
      public String toString()
      { return currentPhrase; }
}
```

Write the Phrase method replaceAllOccurrences, which will replace each occurrence of the string str with the string repl. If no occurrences exist, currentPhrase remains unchanged.

Several examples of the behavior of the method replaceAllOccurrences are shown below.

| | |
|---|---|
| `Phrase phrase1 = new Phrase("A cat ate.");`<br>`phrase1.replaceAllOccurrences("ate", "slept");`<br>`System.out.println(phrase1);` | `A cat slept.` |
| `Phrase phrase2 = new Phrase("aaaa");`<br>`Phrase2.replaceAllOccurrences("aa", "xx");`<br>`System.out.println(phrase4);` | `xxxx` |
| `Phrase phrase3 = new Phrase("aaaa");`<br>`Phrase3.replaceAllOccurrences("aa", "bbb");`<br>`System.out.println(phrase5);` | `bbbbbb` |

Complete method `replaceAllOccurrences` below.

/** Modifies the current phrase by replacing each occurrence of str with repl.
* If there is no occurrence of str, the current phrase is unchanged.
* **Precondition:** str.length() > 0 and n > 0 */

```
public void replaceAllOccurrences(String str, int n, String repl)
```