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 ]
```

(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.

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 12.

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)
```

Part (b) begins on page 13.

AP® COMPUTER SCIENCE A
2015 CANONICAL SOLUTIONS

Question 3: Sparse Array

Part (a):

```
public int getValueAt(int row, int col){
       for (SparseArrayEntry e : entries){
          if (e.getRow() == row && e.getCol() == col){
             return e.getValue();
          }
       }
       return 0;
}
```

Part (b):

```
public void removeColumn(int col){
       int i=0;
       while (i < entries.size()){
            SparseArrayEntry e = entries.get(i);
            if (e.getCol() == col){
                entries.remove(i);
            } else if (e.getCol() > col){
                entries.set(i, new SparseArrayEntry(e.getRow(),
                                                    e.getCol()-1,
                                                    e.getValue()));
                i++;
            } else {
                i++;
            }
      }
      numCols--;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.