Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times, or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

(v) Array/collection access confusion ([ ] get)
(w) Extraneous code that causes side effect (e.g., writing to output, failure to compile)
(x) Local variables used but none declared
(y) Destruction of persistent data (e.g., changing value referenced by parameter)
(z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side effect (e.g., precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- private or public qualifier on a local variable
- Missing public qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- [] vs. () vs. <>
- = instead of == and vice versa
- length/size confusion for array, String, List, or ArrayList, with or without ( )
- Extraneous [] when referencing entire array
- [i,j] instead of [i][j]
- Extraneous size in array declaration (e.g., int[size] nums = new int[size];)
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around if or while conditions

*Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be unambiguously inferred from context; for example, “ArayList” instead of “ArrayList”. As a counterexample, note that if the code declares “Bug bug;”, then uses “Bug.move()” instead of “bug.move()”, the context does not allow for the reader to assume the object instead of the class.
# Question 3: Sparse Array

## Part (a) getValueAt

**Intent:** Return the value at row index \( \text{row} \) and column index \( \text{col} \) in sparse array

+1 Accesses all necessary elements of entries (No bounds errors)

+1 Identifies element of entries at row index \( \text{row} \) and column index \( \text{col} \), if exists

+1 Returns identified value or returns 0 if no entry exists in entries with row index \( \text{row} \) and column index \( \text{col} \)

## Part (b) removeColumn

**Intent:** Remove column \( \text{col} \) from sparse array

+1 Decrements \( \text{numCols} \) exactly once

+1 Accesses all elements of entries (No bounds errors)

+1 Identifies and removes entry with column index \( \text{col} \)

+2 Process entries with column index > \( \text{col} \) within loop

  +1 Creates new \( \text{SparseArrayEntry} \) with current row index, column index -1, current value

  +1 Identifies and replaces entry with column index > \( \text{col} \) with created entry

+1 On exit: All and only entries with column index \( \text{col} \) have been removed and all and only entries with column index > \( \text{col} \) have been changed to have column index -1. All other entries are unchanged. (Minor loop errors ok)

## Question-Specific Penalties

-2 (t) Consistently uses incorrect name instead of entries

-1 (u) Directly accesses private instance variables in \( \text{SparseArrayEntry} \) object
Question 3: Sparse Array

Part (a):

```java
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):

```java
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--;
}
```
Complete method `getValueAt` below.

```java
/**
 * Returns the value of the element at row index `row` and column index `col`
 * in the sparse array.
 * Precondition: 0 ≤ row < numRows()
 * 0 ≤ col < numCols()
 */
public int getValueAt(int row, int col)
    for (SparseArrayEntry n : entries)
        if (n.getRow() == row && n.getCol() == col)
            num = n.getValue();
    return num;
```
Complete method `removeColumn` below.

```java
/**
 * Removes the column `col` from the sparse array.
 * Precondition: 0 ≤ col < getNumCols()
 */
public void removeColumn(int col)
{
    int i = 0;
    while (i < entries.size())
    {
        if ((entries.get(i)).getCol() == col)
        {
            entries.remove(i); // entry in col removed
        }
        else if ((entries.get(i)).getCol() > col)
        {
            SparseArrayEntry k = new SparseArrayEntry(1,
                entries.get(i).getRow(),
                entries.get(i).getCol() - 1,
                entries.get(i).getValue());
            entries.set(i, k);
            i++; // if entry's col > col
        }
        else if ((entries.get(i)).getCol() < col)
        {
            i++;
        }
        // if entry's col < col, no moves.
    }
}
```

Unauthorized copying or reuse of any part of this page is illegal.
Complete method `getValueAt` below.

```java
/**
 * 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){
    for (SparseArrayEntry entry : entries){
        if (entry.getRow() == row && entry.getCol() == col){
            return entry.getValue();
        }
    }
    return 0;
}
```

Part (b) begins on page 16.
Complete method `removeColumn` below.

```java
/** Removes the column `col` from the sparse array.
 * Precondition: 0 ≤ col < getNumCols()
 */
public void removeColumn(int col) {
    for (int idx = 0; idx < entries.size(); idx++) {
        if (entries.get(idx).getCol() == col) {
            entries.remove(idx);
        }
    }
    for (int idx = 0; idx < entries.size(); idx++) {
        if (entries.get(idx).getCol() > col) {
            entries.add(new SparseArray.Entry(entries.get(idx).getRow(),
                                               entries.get(idx).getCol() - 1,
                                               entries.get(idx).getValue()));
            entries.remove(idx);
        }
    }
    self.numCols -= 1;
}
```
Complete method `getValueAt` below.

```java
/** 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)
{
    for (int i = 0; i < entries.size() - 1; i++)
    {  
        if (row == entries[i].getRow() && col == entries[i].getColumn())
            return entries[i].getValue();
        return 0;
    }
}
```

Part (b) begins on page 16.
Complete method `removeColumn` below.

```java
/** Removes the column `col` from the sparse array.
 * Precondition: 0 ≤ col < getNumCols(). */
public void removeColumn(int col)
{
    for (int i = 0; i < entries.size(); i++)
    {
        if (entries[i].getCol() == col)
        {
            entries.remove(i);
        } else if (entries[i].getCol() > col)
        {
            entries.set(i, SparseArrayEntry(entries[i].getRow(), entries[i].getCol() - 1, entries[i].getValue()));
        }
    }
    entries.remove(entries.size() - 1);
}
```
Question 3

Overview

This question involved the use of the `ArrayList` data structure, `ArrayList` traversal, and both the access and modification of `ArrayList` elements. The question also involved modification of a private instance variable. Students were provided with the specifications of two classes: `SparseArrayEntry` and `SparseArray`. The `SparseArray` class represents a sparse array. It contains a list, `entries`, of `SparseArrayEntry` objects that represent non-zero elements in a sparse array. Students were required to call methods from `SparseArrayEntry`.

In part (a) students were asked to write the `SparseArray` method `getValueAt(int row, int col)`, which returns the value of the sparse array element at the given row and column. If no such element exists in the `entries` `ArrayList`, then the method returns 0. In part (b) students were asked to write the `SparseArray` method `removeColumn(int col)` to accomplish three tasks. First, the method removes all elements in `entries` with column index equal to the `col` parameter. Second, the method replaces all elements having a column index greater than the `col` parameter with elements having column indexes decremented by one. Finally, the method decrements the `numCols` instance variable by one to reflect the new dimension of the sparse array.

Sample: 3A
Score: 8

In part (a) the student declares and initializes a local variable `num` to store the result of the method. The student correctly uses an enhanced `for` loop to access all elements of `entries`. The entry at row index `row` and column index `col` is identified correctly using the `SparseArrayEntry` `getRow` and `getCol` methods on each element in `entries`. If an entry with row index `row` and column index `col` is found, the student then correctly uses the `SparseArrayEntry` `getValue` method to assign the identified value to `num`. The method returns the value in `num`, which will be the identified value, or 0 if no entry exists in `entries` with row index `row` and column index `col`. Part (a) earned 3 points.

In part (b), the student does not attempt to decrement the `numCols` instance variable. Therefore, the student does not earn the “Decrement `numCols` Exactly Once” point. The student declares and initializes a loop control variable. The student correctly uses a while loop, combined with the `ArrayList` get method and, therefore, earned the “Access All Elements of `entries`” point. Although no parentheses are used with the call to `size`, there is no penalty as stated in the General Scoring Guidelines. Using an `if` statement, the student correctly identifies elements from the `entries` `ArrayList` whose `col` value is equal to the parameter `col`, then correctly uses the `ArrayList remove method to remove identified elements from entries`. Therefore, the student earned the “Identifies and Removes” point. In an `else-if` statement, the student correctly identifies elements from the `entries` `ArrayList` whose `col` value is greater than the parameter `col`. For each identified element, a new `SparseArrayEntry` object is created by calling the constructor with the current row index, the column index–1, and the current value as parameters. This earned the “Creates New `SparseArrayEntry`” point. The student uses the `ArrayList set method to replace the existing element with the newly created object in the `entries` `ArrayList`. Therefore, the student earned the “Identifies and Replaces” point. The student then correctly increments the loop control variable. The last `else-if` statement is used to correctly increment the loop control variable if the column index is less than `col`, and no change to the element is made. Therefore, the student earned the “On Exit” point. Part (b) earned 5 points.
Sample: 3B
Score: 6

In part (a) the student correctly uses an enhanced for loop to access all elements of entries. The entry at row index \( \text{row} \) and column index \( \text{col} \) is identified correctly using the SparseArrayEntry getRow and getCol methods on each element in entries. If an entry with row index \( \text{row} \) and column index \( \text{col} \) is found, the student then correctly uses the SparseArrayEntry getValue method to return the identified value. Otherwise, the method returns 0 if no entry exists in entries with row index \( \text{row} \) and column index \( \text{col} \). Part (a) earned 3 points.

In part (b) the student correctly uses an indexed for loop, combined with the ArrayList get method, to access elements of entries. Using an if statement, the student correctly identifies elements from the entries ArrayList whose \( \text{col} \) value is equal to the parameter \( \text{col} \). Although no parentheses are used with the call to getCol, there is no penalty as stated in the General Scoring Guidelines. The student then correctly uses the ArrayList remove method to remove identified elements from entries to earn the “Identifies and Removes Entry” point. The student does not decrement the loop control variable resulting in skipped elements within entries. Therefore, the student does not earn the “Access All Elements of entries” point. Within a second for loop, the student uses an if statement to correctly identify elements from the entries ArrayList whose \( \text{col} \) value is greater than the parameter \( \text{col} \). For each identified element, a new SparseArrayEntry object is created by calling the constructor with the current row index, the column index–1, and the current value as parameters. This earned the “Creates New SparseArrayEntry” point. The student uses the ArrayList add method to add the newly created object to the end of the entries ArrayList. The student then uses the ArrayList remove method to remove the identified element from entries. The student earned the “Identifies and Replaces” point. However, because the student adds the modified elements to the end of entries, these new elements will be accessed, and possibly modified, repeatedly. Therefore, the student does not earn the “On Exit” point. The student incorrectly tries to decrement numCols and does not earn the “Decrements numCols Exactly Once” point. Part (b) earned 3 points.

Sample: 3C
Score: 3

In part (a) the student attempts to use an indexed for loop, combined with array notation (see note below), to access elements of entries. However, the conditional check is incorrect, which will result in a failure to check the last element of entries. Therefore, the student does not earn the “Access All Necessary Elements” point. The entry at row index \( \text{row} \) and column index \( \text{col} \) is identified correctly using the SparseArrayEntry getRow and getCol methods on an element in entries. This earned the “Identifies Element of entries” point. If an entry with row index \( \text{row} \) and column index \( \text{col} \) is found, the student then correctly uses the SparseArrayEntry getValue method to return the identified value. Otherwise, the method returns 0 if no entry exists in entries with row index \( \text{row} \) and column index \( \text{col} \). Therefore, the student earned the “Returns Identified Value” point. Part (a) earned 2 points.

In part (b) the student does not attempt to decrement the numCols instance variable. Therefore, the student does not earn the “Decrements numCols Exactly Once” point. The student correctly uses an indexed for loop, combined with array notation (see note below), to access elements of entries. Using an if statement, the student correctly identifies elements from the entries ArrayList whose \( \text{col} \) value is equal to the parameter \( \text{col} \). The student then correctly uses the ArrayList remove method to remove identified elements from entries to earn the “Identifies and Removes Entry” point. The student does not decrement the loop control variable resulting in skipped elements within entries. Therefore, the student
Question 3 (continued)

student does not earn the “Access All Elements of entries” point. In an else-if statement, the student correctly identifies elements from the entries ArrayList whose col value is greater than the parameter col. For each identified element, the student attempts to make a new SparseArrayEntry object by calling the constructor with the current row index, the column index–1, and the current value as parameters. However, the keyword new is not present, so the “Creates New SparseArrayEntry” point is not earned. The student uses the ArrayList set method to replace the existing element with the newly created object in the entries ArrayList. Therefore, the student earned the “Identifies and Replaces” point. Upon completion of the loop, the student removes the last element in entries. Because this element should not have been removed, the student does not earn the “On Exit” point. Part (b) earned 2 points.

Note: The student incorrectly uses array notation in place of the ArrayList get method. This results in a 1-point penalty from the total score as stated in the General Scoring Guidelines. Therefore, the total score is 3.