

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
    /** Initialized in the constructor and contains no null elements */
    private ArrayList<String> wordList;

    /**
     * Returns true if each element of wordList (except the first) contains the previous
     * element as a substring and returns false otherwise, as described in part (a)
     * Precondition: wordList contains at least two elements.
     * Postcondition: wordList is unchanged.
     */
    public boolean isWordChain()
    { /* to be implemented in part (a) */ }

    /**
     * Returns an ArrayList<String> based on strings from wordList that start
     * with target, as described in part (b). Each element of the returned ArrayList has had
     * the initial occurrence of target removed.
     * Postconditions: wordList is unchanged.
     * Items appear in the returned list in the same order as they appear in wordList.
     */
    public ArrayList<String> createList(String target)
    { /* to be implemented in part (b) */ }

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

- (a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

wordList	isWordChain Return Value	Explanation
["an", "band", "band", "abandon"]	true	Each element contains the previous element as a substring.
["to", "too", "stool", "tools"]	false	"tools" does not contain the substring "stool".

Complete the `isWordChain` method.

```
/**
 * Returns true if each element of wordList (except the first) contains the previous
 * element as a substring and returns false otherwise, as described in part (a)
 * Precondition: wordList contains at least two elements.
 * Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

- (b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	ArrayList Returned by <code>createList</code>	Explanation
<code>createList("cat")</code>	<code>["ch", "chacat", ""]</code>	Only "catch", "catchacat", and "cat" begin with "cat".
<code>createList("catch")</code>	<code>["", "acat"]</code>	Only "catch" and "catchacat" begin with "catch".
<code>createList("dog")</code>	<code>[]</code>	None of the words in <code>wordList</code> begin with "dog".

Complete the `createList` method.

```
/**
 * Returns an ArrayList<String> based on strings from wordList that start
 * with target, as described in part (b). Each element of the returned ArrayList has had
 * the initial occurrence of target removed.
 * Postconditions: wordList is unchanged.
 * Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordChecker
private ArrayList<String> wordList
public boolean isWordChain()
public ArrayList<String> createList(String target)
```

3. This question involves the analysis of weather data. The following `WeatherData` class has an instance variable, `temperatures`, which contains the daily high temperatures recorded on consecutive days at a particular location. The class also contains methods used to analyze that data. You will write two methods of the `WeatherData` class.

```
public class WeatherData
{
    /** Guaranteed not to be null and to contain only non-null entries */
    private ArrayList<Double> temperatures;

    /**
     * Cleans the data by removing from temperatures all values that are less than
     * lower and all values that are greater than upper, as described in part (a)
     */
    public void cleanData(double lower, double upper)
    { /* to be implemented in part (a) */ }

    /**
     * Returns the length of the longest heat wave found in temperatures, as described in
     * part (b)
     * Precondition: There is at least one heat wave in temperatures based on threshold.
     */
    public int longestHeatWave(double threshold)
    { /* to be implemented in part (b) */ }

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

- (a) Write the `cleanData` method, which modifies the `temperatures` instance variable by removing all values that are less than the `lower` parameter and all values that are greater than the `upper` parameter. The order of the remaining values in `temperatures` must be maintained.

For example, consider a `WeatherData` object for which `temperatures` contains the following.

99.1	142.0	85.0	85.1	84.6	94.3	124.9	98.0	101.0	102.5
------	-------	------	------	------	------	-------	------	-------	-------

The three shaded values shown would be removed by the method call `cleanData(85.0, 120.0)`.

99.1	142.0	85.0	85.1	84.6	94.3	124.9	98.0	101.0	102.5
------	-------	------	------	------	------	-------	------	-------	-------

The following shows the contents of `temperatures` after the three shaded values are removed as a result of the method call `cleanData(85.0, 120.0)`.

99.1	85.0	85.1	94.3	98.0	101.0	102.5
------	------	------	------	------	-------	-------

Complete method `cleanData`.

```
/**
 * Cleans the data by removing from temperatures all values that are less than
 * lower and all values that are greater than upper, as described in part (a)
 */
public void cleanData(double lower, double upper)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

- (b) Write the `longestHeatWave` method, which returns the length of the longest heat wave found in the `temperatures` instance variable. A heat wave is a sequence of two or more consecutive days with a daily high temperature greater than the parameter `threshold`. The `temperatures` instance variable is guaranteed to contain at least one heat wave based on the `threshold` parameter.

For example, consider the following contents of `temperatures`.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2
-------	------	-------	-------	------	-------	------	------	-------	-------	-------	------

In the following sample contents of `temperatures`, all heat waves based on the threshold temperature of 100.5 are shaded. The method call `longestHeatWave(100.5)` would return 3, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2
-------	------	-------	-------	------	-------	------	------	-------	-------	-------	------

In the following sample contents of `temperatures`, all heat waves based on the threshold temperature of 95.2 are shaded. The method call `longestHeatWave(95.2)` would return 4, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2
-------	------	-------	-------	------	-------	------	------	-------	-------	-------	------

Complete method `longestHeatWave`.

```
/**
 * Returns the length of the longest heat wave found in temperatures, as described in
 * part (b)
 * Precondition: There is at least one heat wave in temperatures based on threshold.
 */
public int longestHeatWave(double threshold)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WeatherData
private ArrayList<Double> temperatures
public void cleanData(double lower, double upper)
public int longestHeatWave(double threshold)
```

3. Many encoded strings contain *delimiters*. A delimiter is a non-empty string that acts as a boundary between different parts of a larger string. The delimiters involved in this question occur in pairs that must be *balanced*, with each pair having an open delimiter and a close delimiter. There will be only one type of delimiter for each string. The following are examples of delimiters.

Example 1

Expressions in mathematics use open parentheses " (" and close parentheses ") " as delimiters. For each open parenthesis, there must be a matching close parenthesis.

$(x + y) * 5$ is a valid mathematical expression.

$(x + (y)$ is NOT a valid mathematical expression because there are more open delimiters than close delimiters.

Example 2

HTML uses `<B>` and `</B>` as delimiters. For each open delimiter `<B>`, there must be a matching close delimiter `</B>`.

`<B> Make this text bold </B>` is valid HTML.

`<B> Make this text bold </UB>` is NOT valid HTML because there is one open delimiter and no matching close delimiter.

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

    /** The open and close delimiters. */
    private String openDel;
    private String closeDel;

    /** Constructs a Delimiters object where open is the open delimiter and close is the
     * close delimiter.
     * Precondition: open and close are non-empty strings.
     */
    public Delimiters(String open, String close)
    {
        openDel = open;
        closeDel = close;
    }

    /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
    public ArrayList<String> getDelimitersList(String[] tokens)
    { /* to be implemented in part (a) */ }

    /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
     * Precondition: delimiters contains only valid open and close delimiters.
     */
    public boolean isBalanced(ArrayList<String> delimiters)
    { /* to be implemented in part (b) */ }

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

- (a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

openDel: " ("

closeDel: ") "

tokens:

" ("	"x + y"	") "	" * 5"
------	---------	------	--------

ArrayList
of delimiters:

" ("	") "
------	------

Example 2

openDel: "<q>"

closeDel: "</q>"

tokens:

"<q>"	"yy"	"</q>"	"zz"	"</q>"
-------	------	--------	------	--------

ArrayList
of delimiters:

"<q>"	"</q>"	"</q>"
-------	--------	--------

Class information for this question

```
public class Delimiters
private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

- (b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1

The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

"^{"	"^{"	"}"	"^{"	"}"	"}"
---------	---------	----------	---------	----------	----------

Example 2

The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"^{"	"}"	"</sup>"	"<sup>"
---------	----------	----------	---------



When starting from the left, at this point, condition 1 is violated.

Example 3

The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"</sup>"



At this point, condition 1 is violated.

Example 4

The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.

"<sup>"	"^{"	"}"
---------	---------	----------

Class information for this question

```
public class Delimiters
{
    private String openDel;
    private String closeDel;

    public Delimiters(String open, String close)
    {
        openDel = open;
        closeDel = close;
    }

    public ArrayList<String> getDelimitersList(String[] tokens)
    {
        ArrayList<String> delimiters = new ArrayList<>();
        for (String token : tokens)
        {
            delimiters.add(token);
        }
        return delimiters;
    }

    public boolean isBalanced(ArrayList<String> delimiters)
    {
        // Returns true if the delimiters are balanced and false otherwise, as described in part (b).
        // Precondition: delimiters contains only valid open and close delimiters.
    }
}
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 * Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
{
    // Complete the method
}
```

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{
    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    { /* implementation not shown */ }

    /** Returns the first string of this WordPair object. */
    public String getFirst()
    { /* implementation not shown */ }

    /** Returns the second string of this WordPair object. */
    public String getSecond()
    { /* implementation not shown */ }
}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{
    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;

    /** Constructs a WordPairList object as described in part (a).
     *   Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }

    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }
}
```

- (a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects. A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects (`words[i]`, `words[j]`) for every `i` and `j`, where $0 \leq i < j < \text{words.length}$. Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair
{
    public WordPair(String first, String second)
    public String getFirst()
    public String getSecond()

    public class WordPairList
    {
        private ArrayList<WordPair> allPairs

        public WordPairList(String[] words)
        public int numMatches()
    }
}
```

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
```

- (b) Write the `WordPairList` method `numMatches`. This method returns the number of `WordPair` objects in `allPairs` for which the two strings match.

For example, the following code segment creates a `WordPairList` object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return 2.

Class information for this question

```
public class WordPair
{
    public WordPair(String first, String second)
    public String getFirst()
    public String getSecond()
}

public class WordPairList
{
    private ArrayList<WordPair> allPairs

    public WordPairList(String[] words)
    public int numMatches()
}
```

Complete method `numMatches` below.

```
/** Returns the number of matches as described in part (b).
 *
 */
```

1. This question involves identifying and processing the digits of a non-negative integer. The declaration of the `Digits` class is shown below. You will write the constructor and one method for the `Digits` class.

```
public class Digits
{
    /** The list of digits from the number used to construct this object.
     * The digits appear in the list in the same order in which they appear in the original number.
     */
    private ArrayList<Integer> digitList;

    /** Constructs a Digits object that represents num.
     * Precondition: num >= 0
     */
    public Digits(int num)
    { /* to be implemented in part (a) */ }

    /** Returns true if the digits in this Digits object are in strictly increasing order;
     * false otherwise.
     */
    public boolean isStrictlyIncreasing()
    { /* to be implemented in part (b) */ }
}
```

Part (a) begins on page 4.

- (a) Write the constructor for the `Digits` class. The constructor initializes and fills `digitList` with the digits from the non-negative integer `num`. The elements in `digitList` must be `Integer` objects representing single digits, and appear in the same order as the digits in `num`. Each of the following examples shows the declaration of a `Digits` object and the contents of `digitList` as initialized by the constructor.

Example 1

```
Digits d1 = new Digits(15704);
```

d1:

	0	1	2	3	4
digitList:	1	5	7	0	4

Example 2

```
Digits d2 = new Digits(0);
```

d2:

	0
digitList:	0

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete the `Digits` constructor below.

```
/** Constructs a Digits object that represents num.
 * Precondition: num >= 0
 */
public Digits(int num)
```

Part (b) begins on page 6.

- (b) Write the `Digits` method `isStrictlyIncreasing`. The method returns `true` if the elements of `digitList` appear in strictly increasing order; otherwise, it returns `false`. A list is considered strictly increasing if each element after the first is greater than (but not equal to) the preceding element.

The following table shows the results of several calls to `isStrictlyIncreasing`.

Method call	Value returned
<code>new Digits(7).isStrictlyIncreasing()</code>	<code>true</code>
<code>new Digits(1356).isStrictlyIncreasing()</code>	<code>true</code>
<code>new Digits(1336).isStrictlyIncreasing()</code>	<code>false</code>
<code>new Digits(1536).isStrictlyIncreasing()</code>	<code>false</code>
<code>new Digits(65310).isStrictlyIncreasing()</code>	<code>false</code>

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `isStrictlyIncreasing` below.

```
/** Returns true if the digits in this Digits object are in strictly increasing order;
 *      false otherwise.
 */
```

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width*. Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

Total number of letters in words: 14
 Number of gaps between words: 3
 Basic gap width: 2
 Leftover spaces: 0

Formatted string:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
| A | P |   |   | C | O | M | P |   |   | S | C | I |   |   | R | O | C | K | S |
```

Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

Total number of letters in words: 15
 Number of gaps between words: 3
 Basic gap width: 1
 Leftover spaces: 2

The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

Formatted string:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
| G | R | E | E | N |   |   | E | G | G | S |   |   | A | N | D |   | H | A | M |
```

Example 3: `wordList: ["BEACH", "BALL"]`

Total number of letters in words: 9
 Number of gaps between words: 1
 Basic gap width: 11
 Leftover spaces: 0

Formatted string:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
| B | E | A | C | H |   |   |   |   |   |   |   |   |   |   | B | A | L | L |
```

You will implement three `static` methods in a class named `StringFormatter` that is not shown.

- (a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns 7. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 * Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

Part (b) begins on page 20.

- (b) Write the `StringFormatter` method `basicGapWidth`, which returns the basic gap width as defined earlier.

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 * a formatted string of formattedLen characters.
 * Precondition: wordList contains at least two words, consisting of letters only.
 * formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
```

Part (c) begins on page 22.

- (c) Write the `StringFormatter` method `format`, which returns the formatted string as defined earlier. The `StringFormatter` class also contains a method called `leftoverSpaces`, which has already been implemented. This method returns the number of leftover spaces as defined earlier and is shown below.

```
/** Returns the number of leftover spaces when wordList is used to produce
 * a formatted string of formattedLen characters.
 * Precondition: wordList contains at least two words, consisting of letters only.
 * formattedLen is large enough for all the words and gaps.
 */
public static int leftoverSpaces(List<String> wordList,
                                int formattedLen)
{ /* implementation not shown */ }
```

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 * Precondition: The wordList contains at least two words, consisting of letters only.
 *               formattedLen is large enough for all the words and gaps.
 * Postcondition: All words in wordList appear in the formatted string.
 *   - The words appear in the same order as in wordList.
 *   - The number of spaces between words is determined by basicGapWidth and the
 *     distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

STOP

END OF EXAM

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 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:  $0 \leq \text{row} < \text{getNumRows}()$ 
     *  $0 \leq \text{col} < \text{getNumCols}()$ 
     */
    public int getValueAt(int row, int col)
    { /* to be implemented in part (a) */ }

    /** Removes the column col from the sparse array.
     * Precondition:  $0 \leq \text{col} < \text{getNumCols}()$ 
     */
    public void removeColumn(int col)
    { /* 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.

	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:  $0 \leq \text{row} < \text{getNumRows}()$ 
 *                  $0 \leq \text{col} < \text{getNumCols}()$ 
 */
public int getValueAt(int row, int col)
```

Part (b) begins on page 13.

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

- All entries in the list `entries` with column indexes matching `col` are removed from the list.
- All entries in the list `entries` with column indexes greater than `col` are replaced by entries with column indexes that are decremented by one (moved one column to the left).
- The number of columns in the sparse array is adjusted to reflect the column 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 shaded entries in `entries`, below, correspond to the shaded column above.

`numRows:` 6

`numCols:` 5

<code>entries:</code>	row: 1 col: 4 value: 4	row: 2 col: 0 value: 1	row: 3 col: 1 value: -9	row: 1 col: 1 value: 5
-----------------------	---	---	--	---

When `sparse` has the state shown above, the call `sparse.removeColumn(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:` 6

`numCols:` 4

<code>entries:</code>	row: 1 col: 3 value: 4	row: 2 col: 0 value: 1
-----------------------	---	---

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 removeColumn(int col)
```

Complete method `removeColumn` below.

```
/** Removes the column col from the sparse array.
 * Precondition:  $0 \leq \text{col} < \text{getNumCols}()$ 
 */
public void removeColumn(int col)
```

4. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

- (a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write an interface named `NumberGroup` that represents a group of integers. The interface should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

Write the complete `NumberGroup` interface. It must have exactly one method.

Part (b) begins on page 17.

- (b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.

Write the `Range` class, which is a `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

Part (c) begins on page 18.

- (c) The `MultipleGroups` class (not shown) represents a collection of `NumberGroup` objects and is a `NumberGroup`. The `MultipleGroups` class stores the number groups in the instance variable `groupList` (shown below), which is initialized in the constructor.

```
private List<NumberGroup> groupList;
```

Write the `MultipleGroups` method `contains`. The method takes an integer and returns `true` if and only if the integer is contained in one or more of the number groups in `groupList`.

For example, suppose `multiple1` has been declared as an instance of `MultipleGroups` and consists of the three ranges created by the calls `new Range(5, 8)`, `new Range(10, 12)`, and `new Range(1, 6)`. The following table shows the results of several calls to `contains`.

Call	Result
<code>multiple1.contains(2)</code>	<code>true</code>
<code>multiple1.contains(9)</code>	<code>false</code>
<code>multiple1.contains(6)</code>	<code>true</code>

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *     false otherwise.
 */
public boolean contains(int num)
```

STOP
END OF EXAM

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the appendices have been imported where appropriate.
 - Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
 - In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.
1. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.
- (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.
- The scrambling process begins at the first letter of the word and continues from left to right.
 - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
 - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

The following table shows several examples of words and their scrambled versions.

word	Result returned by <code>scrambleWord(word)</code>
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A "	"A "
""	""

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 * @param word the word to be scrambled
 * @return the scrambled word (possibly equal to word)
 * Precondition: word is either an empty string or contains only uppercase letters.
 * Postcondition: the string returned was created from word as follows:
 *   - the word was scrambled, beginning at the first letter and continuing from left to right
 *   - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *   - letters were swapped at most once
 */
public static String scrambleWord(String word)
```

Part (b) begins on page 4.

- (b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
wordList	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
wordList	"TNA"	"BARCADABARA"	"PAPLE"

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 * version, removing any words that are unchanged as a result of scrambling.
 * @param wordList the list of words
 * Precondition: wordList contains only non-null objects
 * Postcondition:
 *   - all words unchanged by scrambling have been removed from wordList
 *   - each of the remaining words has been replaced by its scrambled version
 *   - the relative ordering of the entries in wordList is the same as it was
 *     before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```

3. A student in a school is represented by the following class.

```
public class Student
{
    /** Returns the name of this Student. */
    public String getName()
    { /* implementation not shown */ }

    /** Returns the number of times this Student has missed class. */
    public int getAbsenceCount()
    { /* implementation not shown */ }

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

The class `SeatingChart`, shown below, uses a two-dimensional array to represent the seating arrangement of students in a classroom. The seats in the classroom are in a rectangular arrangement of rows and columns.

```
public class SeatingChart
{
    /** seats[r][c] represents the Student in row r and column c in the classroom. */
    private Student[][] seats;

    /** Creates a seating chart with the given number of rows and columns from the students in
     *  studentList. Empty seats in the seating chart are represented by null.
     *  @param rows the number of rows of seats in the classroom
     *  @param cols the number of columns of seats in the classroom
     *  Precondition: rows > 0; cols > 0;
     *                  rows * cols >= studentList.size()
     *  Postcondition:
     *  - Students appear in the seating chart in the same order as they appear
     *    in studentList, starting at seats[0][0].
     *  - seats is filled column by column from studentList, followed by any
     *    empty seats (represented by null).
     *  - studentList is unchanged.
     */
    public SeatingChart(List<Student> studentList,
                       int rows, int cols)
    { /* to be implemented in part (a) */ }

    /** Removes students who have more than a given number of absences from the
     *  seating chart, replacing those entries in the seating chart with null
     *  and returns the number of students removed.
     *  @param allowedAbsences an integer >= 0
     *  @return number of students removed from seats
     *  Postcondition:
     *  - All students with allowedAbsences or fewer are in their original positions in seats.
     *  - No student in seats has more than allowedAbsences absences.
     *  - Entries without students contain null.
     */
    public int removeAbsentStudents(int allowedAbsences)
    { /* to be implemented in part (b) */ }

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

- (a) Write the constructor for the `SeatingChart` class. The constructor initializes the `seats` instance variable to a two-dimensional array with the given number of rows and columns. The students in `studentList` are copied into the seating chart in the order in which they appear in `studentList`. The students are assigned to consecutive locations in the array `seats`, starting at `seats[0][0]` and filling the array column by column. Empty seats in the seating chart are represented by `null`.

For example, suppose a variable `List<Student> roster` contains references to `Student` objects in the following order.

"Karen" 3	"Liz" 1	"Paul" 4	"Lester" 1	"Henry" 5	"Renee" 9	"Glen" 2	"Fran" 6	"David" 1	"Danny" 3
--------------	------------	-------------	---------------	--------------	--------------	-------------	-------------	--------------	--------------

A `SeatingChart` object created with the call `new SeatingChart(roster, 3, 4)` would have `seats` initialized with the following values.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

Complete the `SeatingChart` constructor below.

```
/** Creates a seating chart with the given number of rows and columns from the students in
 *  studentList. Empty seats in the seating chart are represented by null.
 *  @param rows the number of rows of seats in the classroom
 *  @param cols the number of columns of seats in the classroom
 *  Precondition: rows > 0; cols > 0;
 *                  rows * cols >= studentList.size()
 *  Postcondition:
 *    - Students appear in the seating chart in the same order as they appear
 *      in studentList, starting at seats[0][0].
 *    - seats is filled column by column from studentList, followed by any
 *      empty seats (represented by null).
 *    - studentList is unchanged.
 */
public SeatingChart(List<Student> studentList,
                   int rows, int cols)
```

Part (b) begins on page 10.

- (b) Write the `removeAbsentStudents` method, which removes students who have more than a given number of absences from the seating chart and returns the number of students that were removed. When a student is removed from the seating chart, a `null` is placed in the entry for that student in the array `seats`. For example, suppose the variable `SeatingChart introCS` has been created such that the array `seats` contains the following entries showing both students and their number of absences.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

After the call `introCS.removeAbsentStudents(4)` has executed, the array `seats` would contain the following values and the method would return the value 3.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	null	null	null
2	"Paul" 4	null	"David" 1	null

Class information repeated from the beginning of the question:

```
public class Student

public String getName()
public int getAbsenceCount()

public class SeatingChart

private Student[][] seats
public SeatingChart(List<Student> studentList,
                   int rows, int cols)
public int removeAbsentStudents(int allowedAbsences)
```

Complete method `removeAbsentStudents` below.

```
/** Removes students who have more than a given number of absences from the
 * seating chart, replacing those entries in the seating chart with null
 * and returns the number of students removed.
 * @param allowedAbsences an integer  $\geq 0$ 
 * @return number of students removed from seats
 * Postcondition:
 *   - All students with allowedAbsences or fewer are in their original positions in seats.
 *   - No student in seats has more than allowedAbsences absences.
 *   - Entries without students contain null.
 */
public int removeAbsentStudents(int allowedAbsences)
```