

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (". "), exclamation point ("! "), or letter, or is a `String` of length 0 if the user did not enter a comment.

```
public class Review
{
    private int rating;
    private String comment;

    /** Precondition: r >= 0
     *      c is not null.
     */
    public Review(int r, String c)
    {
        rating = r;
        comment = c;
    }

    public int getRating()
    {
        return rating;
    }

    public String getComment()
    {
        return comment;
    }

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

GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    { /* implementation not shown */ }

    /** Returns a double representing the average rating of all the Review objects to be
     * analyzed, as described in part (a)
     * Precondition: allReviews contains at least one Review.
     * No element of allReviews is null.
     */
    public double getAverageRating()
    { /* to be implemented in part (a) */ }

    /** Returns an ArrayList of String objects containing formatted versions of
     * selected user comments, as described in part (b)
     * Precondition: allReviews contains at least one Review.
     * No element of allReviews is null.
     * Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    { /* to be implemented in part (b) */ }
}
```

GO ON TO THE NEXT PAGE.

- (a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return 3.4 if `allReviews` contained the following `Review` objects.

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 * analyzed, as described in part (a)
 * Precondition: allReviews contains at least one Review.
 * No element of allReviews is null.
 */
public double getAverageRating()
```

**Begin your response at the top of a new page in the Free Response booklet
and fill in the appropriate circle indicating 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 Review
private int rating
private String comment
public Review(int r, String c)
public int getRating()
public String getComment()
public class ReviewAnalysis
private Review[] allReviews
public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (" - ").
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index 1 and index 4 in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.

"0-Good! Thx."	"2-Great!"	"3-Poor! Bad."
----------------	------------	----------------

Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 * selected user comments, as described in part (b)
 * Precondition: allReviews contains at least one Review.
 * No element of allReviews is null.
 * Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

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

GO ON TO THE NEXT PAGE.

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)
```

```
public ArrayList<String> getDelimitersList(String[] tokens)
```

```
public boolean isBalanced(ArrayList<String> 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)
```

COMPUTER SCIENCE A**SECTION II****Time—1 hour and 30 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 Java Quick Reference 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 will not receive full credit.

1. This question involves the implementation and extension of a `RandomStringChooser` class.

- (a) A `RandomStringChooser` object is constructed from an array of non-null `String` values. When the object is first constructed, all of the strings are considered available. The `RandomStringChooser` class has a `getNext` method, which has the following behavior. A call to `getNext` returns a randomly chosen string from the available strings in the object. Once a particular string has been returned from a call to `getNext`, it is no longer available to be returned from subsequent calls to `getNext`. If no strings are available to be returned, `getNext` returns `"NONE"`.

The following code segment shows an example of the behavior of `RandomStringChooser`.

```
String[] wordArray = {"wheels", "on", "the", "bus"};
RandomStringChooser sChooser = new RandomStringChooser(wordArray);
for (int k = 0; k < 6; k++)
{
    System.out.print(sChooser.getNext() + " ");
}
```

One possible output is shown below. Because `sChooser` has only four strings, the string `"NONE"` is printed twice.

```
bus the wheels on NONE NONE
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

Part (b) begins on page 4.

- (b) The following partially completed `RandomLetterChooser` class is a subclass of the `RandomStringChooser` class. You will write the constructor for the `RandomLetterChooser` class.

```
public class RandomLetterChooser extends RandomStringChooser
{
    /** Constructs a random letter chooser using the given string str.
     * Precondition: str contains only letters.
     */
    public RandomLetterChooser(String str)
    { /* to be implemented in part (b) */ }

    /** Returns an array of single-letter strings.
     * Each of these strings consists of a single letter from str. Element k
     * of the returned array contains the single letter at position k of str.
     * For example, getSingleLetters("cat") returns the
     * array { "c", "a", "t" }.
     */
    public static String[] getSingleLetters(String str)
    { /* implementation not shown */ }
}
```

The following code segment shows an example of using `RandomLetterChooser`.

```
RandomLetterChooser letterChooser = new RandomLetterChooser("cat");
for (int k = 0; k < 4; k++)
{
    System.out.print(letterChooser.getNext());
}
```

The code segment will print the three letters in "cat" in one of the possible orders. Because there are only three letters in the original string, the code segment prints "NONE" the fourth time through the loop. One possible output is shown below.

actNONE

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 * Precondition: str contains only letters.
 */
public RandomLetterChooser(String str)
```

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 Java Quick Reference 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 will not receive full credit.
1. This question involves reasoning about one-dimensional and two-dimensional arrays of integers. You will write three static methods, all of which are in a single enclosing class, named `DiverseArray` (not shown). The first method returns the sum of the values of a one-dimensional array; the second method returns an array that represents the sums of the rows of a two-dimensional array; and the third method analyzes row sums.
- (a) Write a static method `arraySum` that calculates and returns the sum of the entries in a specified one-dimensional array. The following example shows an array `arr1` and the value returned by a call to `arraySum`.

<u>arr1</u>					Value returned by <u>arraySum(arr1)</u>
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.  
 */  
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

- (b) Write a static method `rowSums` that calculates the sums of each of the rows in a given two-dimensional array and returns these sums in a one-dimensional array. The method has one parameter, a two-dimensional array `arr2D` of `int` values. The array is in row-major order: `arr2D[r][c]` is the entry at row `r` and column `c`. The method returns a one-dimensional array with one entry for each row of `arr2D` such that each entry is the sum of the corresponding row in `arr2D`. As a reminder, each row of a two-dimensional array is a one-dimensional array.

For example, if `mat1` is the array represented by the following table, the call `rowSums(mat1)` returns the array `{16, 32, 28, 20}`.

	<u>mat1</u>				
	0	1	2	3	4
0	1	3	2	7	3
1	10	10	4	6	2
2	5	3	5	9	6
3	7	6	4	2	1

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 * the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
```

Part (c) begins on page 6.

- (c) A two-dimensional array is *diverse* if no two of its rows have entries that sum to the same value. In the following examples, the array `mat1` is diverse because each row sum is different, but the array `mat2` is not diverse because the first and last rows have the same sum.

	<u>mat1</u>					
	0	1	2	3	4	Row sums
0	1	3	2	7	3	16
1	10	10	4	6	2	32
2	5	3	5	9	6	28
3	7	6	4	2	1	20

	<u>mat2</u>					
	0	1	2	3	4	Row sums
0	1	1	5	3	4	14
1	12	7	6	1	9	35
2	8	11	10	2	5	36
3	3	2	3	0	6	14

Write a static method `isDiverse` that determines whether or not a given two-dimensional array is diverse. The method has one parameter: a two-dimensional array `arr2D` of `int` values. The method should return `true` if all the row sums in the given array are unique; otherwise, it should return `false`. In the arrays shown above, the call `isDiverse(mat1)` returns `true` and the call `isDiverse(mat2)` returns `false`.

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

```
/** Returns true if all rows in arr2D have different row sums;
 *     false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
```