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.

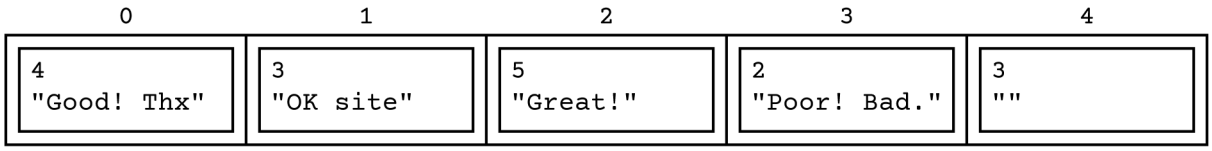
```
public class Review
{
   private int rating;
   private String comment;

   /** 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) */  }
}
```

(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.



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

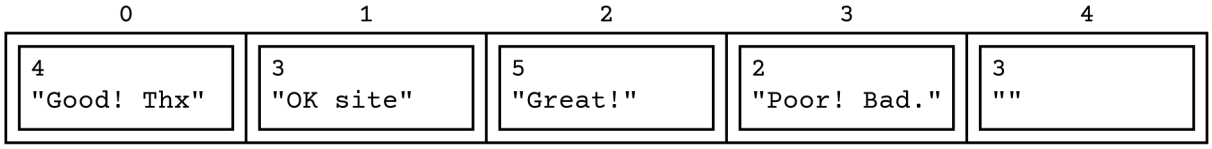
```
public class Review

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.

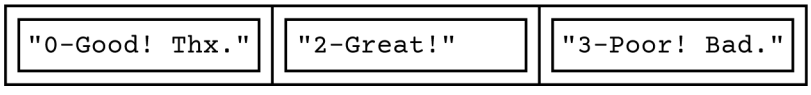
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `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).



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.



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.

4. This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the `Data` class is shown. You will write two methods of the `Data` class.

```
public class Data
{
   public static final int MAX = /* value not shown */;
   private int[][] grid;

   /** Fills all elements of grid with randomly generated values, as described in part (a)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public void repopulate()
   {  /* to be implemented in part (a) */  }

   /** Returns the number of columns in grid that are in increasing order, as described
    *  in part (b)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public int countIncreasingCols()
   {  /* to be implemented in part (b) */  }

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

Question 3: Array / ArrayList — 9 points

Canonical solution

(a) — **3 points**

```
public double getAverageRating()
{
   int sum = 0;

   for (Review r : allReviews)
   {
      sum += r.getRating();
   }

   return (double) sum / allReviews.length;
}
```

(b) — **6 points**

```
public ArrayList<String> collectComments()
{
   ArrayList<String> commentList = new ArrayList<String>();

   for (int i = 0; i < allReviews.length; i++)
   {
      String comment = allReviews[i].getComment();
      if (comment.indexOf("!") >= 0)
      {
         String last =
            comment.substring(comment.length() - 1);
         if (!last.equals("!") && !last.equals("."))
         {
            comment += ".";
         }
         commentList.add(i + "-" + comment);
      }
   }
   return commentList;
}
```

(b) `collectComments`

	Scoring Criteria	Decision Rules	
4	Instantiates an `ArrayList` capable of holding `String` objects		**1 point**
5	Accesses every element of `allReviews` (*no bounds errors*)	Responses **can** still earn the point even if they • fail to keep track of the index Responses **will not** earn the point if they • access the elements of `allReviews` incorrectly	**1 point**
6	Calls `getComment` on an element of `allReviews`, calls at least one `String` method appropriately on the `getComment` return value, and all `String` method calls are syntactically	Responses **can** still earn the point even if they • call some of the `String` methods on objects other than `getComment` return values Responses **will not** earn the point if they • include a parameter when calling `getComment` • call any `String` methods incorrectly • call any `String` methods on objects other than `String` values	**1 point**
7	Compares the final character of the comment to both a period and an exclamation point	Responses **can** still earn the point even if they • use incorrect logic in the comparison • call `String` methods incorrectly Responses **will not** earn the point if they • use `==` instead of `equals` when comparing `String` objects	**1 point**
8	Assembles string appropriately based on result of comparison of last character with period and exclamation point (*algorithm*)	Responses **can** still earn the point even if they • call `String` methods incorrectly • use `==` instead of `equals` Responses **will not** earn the point if they • fail to keep track of the element index • use incorrect logic in the comparison	**1 point**
9	Adds all and only appropriate constructed strings to the `ArrayList` (*algorithm*)	Responses **can** still earn the point even if they • initialize the `ArrayList` incorrectly • fail to return the constructed `ArrayList` (*return is not assessed*) • assemble the review string incorrectly • access the elements of `allReviews` incorrectly	**1 point**

Total for part (b)	**6 points**

Question-specific penalties
None

Total for question 3	**9 points**

Applying the Scoring Criteria

Apply the question scoring criteria 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., printing to output, incorrect precondition check)
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., valid 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 `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*