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.
}
```

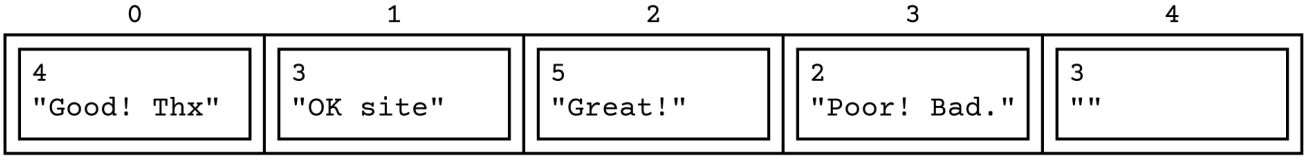
GO ON TO THE NEXT PAGE.

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.



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

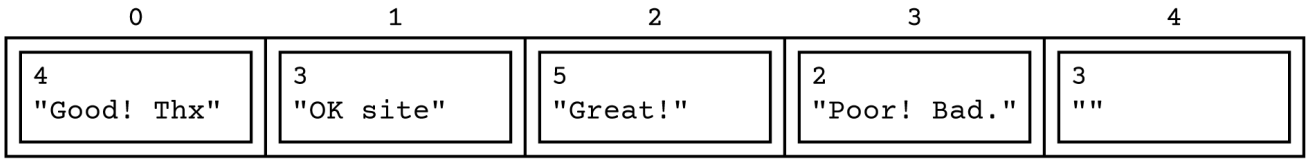
```
public class Review

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.

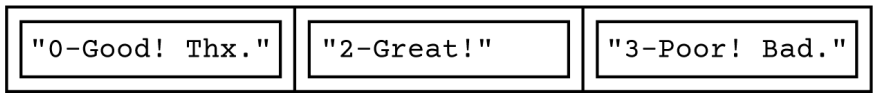
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty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

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)
```

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 $0 \le i < j <$ `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).
 */
```