1.When Web developers want to convert plain text documents to HTML, they need to make changes to the text to insert tags and replace certain characters. For this question, you will write part of the class to convert text to HTML. You will write methods to replace underscores ("_") with tags indicating the text should be in italics.

The class definition for this problem is given below:

```
public class TextFormatter
{
  private String line; //  The line to format

  public TextFormatter (String lineToFormat)
  {  line = lineToFormat;  }


  /**
   * Finds the first single instance of str in line,
   * starting at the position start
   * @param str the string of length 1 to find.
   * Guaranteed to be length 1.
   * @param start the position to start searching.
   * Guaranteed to be in the string line.
   * @return the index of the first single instance of
   * str if the string is found or -1 if it is not found.
   */
  private int findString (String str, int start)
  {  /* To be implemented in part a) */ }


  /**
   * Count the number of times single instances of str appear in
   * the line.
   * @param str the string to find.
   * Guaranteed to be length 1.
   * @return the number of times the string appears in the line
   */
  private int countStrings (String str)
  { /* */ }


  /**
   * Replace all single instances of underscores in the line given by
   * line with italics tags.  There must be an even number of underscores
```

```
    * in the line, and they will be replaced by <I>, </I>, alternating.
    * @param original a string of length 1 to replace
    * @param replacement the string (of any length) use as a replacement
    * @return the line with single instances of underscores replaced with
    * <I> tags or the original line if there are not an even number of
    * underscores.
    */
   public String convertItalics ()
   {  /* To be implemented in part b) */ }
}
```

a) Write the method `findString`. This method will take a string of length 1 to find in the line, at a given starting point. It will return the location of the goal string. This differs from the `indexOf` method of the `String` class because it requires the string be just a single instance of the string, so if the string appears two or more times consecutively, it will not return any of those values. If there is no single instance, the method should return -1. Consider the following examples, where `line` has the value `"aabaccb"`.

Call	Result	Explanation
`findString("a", 0)`	3	The first single occurrence of `"a"` is in position 3. The `"a"`s in position 0 and 1 are not returned because they are not single instances.
`findString("b", 4)`	6	The first single occurrence of `"b"` at or after position 4 is in position 6.
`findString("c", 0)`	-1	There is no single instance of `"c"` in the line.

```
    private int findString (String str, int start){}
```

b) Write the method `convertItalics`. If the line has an even number of single underscores, a line with those underscores converted to alternating <I> and </I> tags should be returned. The first underscore will be converted to <I>, the second to </I>, the third to <I>, and so on. If the line does not have an even number of <I> tags, the original line should be returned. Consider the following examples. Must call the method `countStrings` to get the credits.

```
    public String convertItalics (){}
```

`line`	value returned by `convertItalics`
`This is _very_ good.`	`This is <I>very</I> good.`
`_This_ is _very_ _good_.`	`<I>This</I> is <I>very</I> <I>good</I>.`
`This is _very good.`	`This is _very good.`
`This is __very good.`	`This is __very good.`

2. **SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.**

- 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.

The `Bus` class simulates the activity of a bus. A bus moves back and forth along a single route, making stops along the way. The stops on the route are numbered consecutively starting from 1 up to and including a number that is provided when the `Bus` object is created. You may assume that the number of stops will always be greater than 1.

The bus starts at the first stop and is initially heading toward the last stop. At each step of the simulation, the bus is at a particular stop and is heading toward either the first or last stop. When the bus reaches the first or last stop, it reverses direction. The following table contains a sample code execution sequence and the corresponding results.

Statement or Expression	Value returned (blank if no value)	Comment
`Bus bus1 = new Bus(3);`		The route for `bus1` has three stops numbered 1–3.
`bus1.getCurrentStop();`	1	`bus1` is at stop 1 (first stop on the route).
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.move();`		`bus1` moves to the next stop (3).
`bus1.getCurrentStop();`	3	`bus1` is at stop 3.
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.move();`		`bus1` moves to the next stop (1).
`bus1.move();`		`bus1` moves to the next stop (2).
`bus1.getCurrentStop();`	2	`bus1` is at stop 2.
`bus1.getCurrentStop();`	2	`bus1` is still at stop 2.
`Bus bus2 = new Bus(5);`		The route for `bus2` has five stops numbered 1–5.
`bus1.getCurrentStop();`	2	`bus1` is still at stop 2.
`bus2.getCurrentStop();`	1	`bus2` is at stop 1 (first stop on the route).

Write the complete `Bus` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.

3. A student plans to analyze product reviews found on a Web site by looking for keywords in posted reviews. The `ProductReview` class, shown below, is used to represent a single review. A product review consists of a product name and a review of that product.

```
public class ProductReview
{
   private String name;
   private String review;

   /** Constructs a ProductReview object and initializes the instance variables. */
   public ProductReview(String pName, String pReview)
   {
      name = pName;
      review = pReview;
   }

   /** Returns the name of the product. */
   public String getName()
   {  return name; }

   /** Returns the review of the product. */
   public String getReview()
   {  return review; }
}
```

The `ReviewCollector` class, shown below, is used to represent a collection of reviews to be analyzed.

```
public class ReviewCollector
{
   private ArrayList<ProductReview> reviewList;
   private ArrayList<String> productList;

   /** Constructs a ReviewCollector object and initializes the instance variables. */
   public ReviewCollector()
   {
      reviewList = new ArrayList<ProductReview>();
      productList = new ArrayList<String>();
   }

   /** Adds a new review to the collection of reviews, as described in part (a). */
   public void addReview(ProductReview prodReview)
   {  /* to be implemented in part (a) */  }

   /** Returns the number of good reviews for a given product name, as described in part (b). */
   public int getNumGoodReviews(String prodName)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `addReview` method, which adds a single product review, represented by a `ProductReview` object, to the `ReviewCollector` object. The `addReview` method does the following when it adds a product review.

- The `ProductReview` object is added to the `reviewList` instance variable.
- The product name from the `ProductReview` object is added to the `productList` instance variable if the product name is not already found in `productList`.

Elements may be added to `reviewList` and `productList` in any order.

Complete method `addReview`.

```
/** Adds a new review to the collection of reviews, as described in part (a). */
public void addReview(ProductReview prodReview)
```

(b) Write the `getNumGoodReviews` method, which returns the number of *good* reviews for a given product name. A review is considered good if it contains the string `"best"` (all lowercase). If there are no reviews with a matching product name, the method returns `0`. Note that a review that contains `"BEST"` or `"Best"` is not considered a good review (since not all the letters of `"best"` are lowercase), but a review that contains `"asbestos"` is considered a good review (since all the letters of `"best"` are lowercase).

Complete method `getNumGoodReviews`.

```
/** Returns the number of good reviews for a given product name, as described in part (b). */
public int getNumGoodReviews(String prodName)
```

Class information for this question

```
public class ProductReview

private String name
private String review

public ProductReview(String pName, String pReview)
public String getName()
public String getReview()

public class ReviewCollector

private ArrayList<ProductReview> reviewList
private ArrayList<String> productList

public ReviewCollector()
public void addReview(ProductReview prodReview)
public int getNumGoodReviews(String prodName)
```

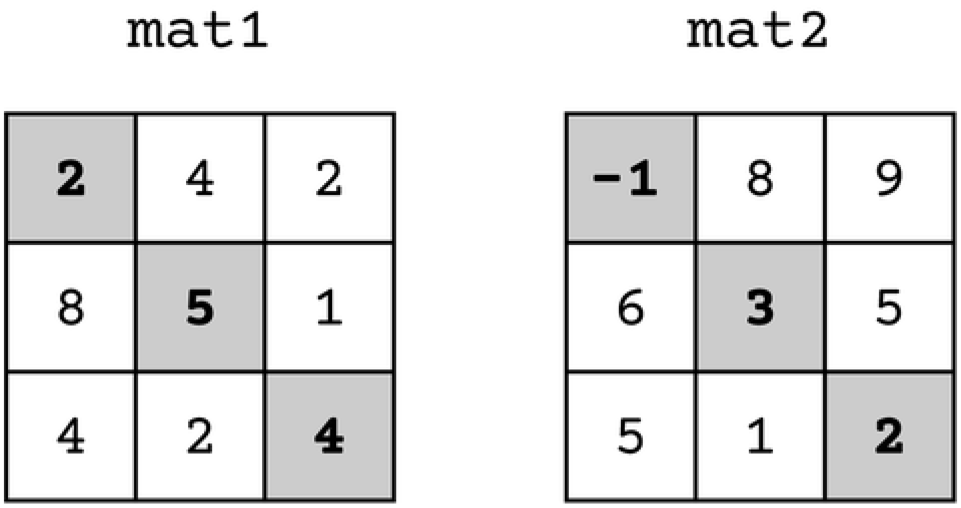
4. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- 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.

This question involves performing arithmetic operations on two-dimensional (2D) arrays of integers. You will write two static methods, both of which are in a class named `MatrixOp` (not shown).

(a) Write the method `diagonalOp`, which returns the sum of the products of the corresponding entries on the main diagonals of two given square 2D arrays that have the same dimensions. The main diagonal goes from the top-left corner to the bottom-right corner in a square 2D array.

For example, assume that `mat1` and `mat2` are properly defined 2D arrays containing the values shown below. The main diagonals have been shaded in gray.



mat1

2	4	2
8	**5**	1
4	2	**4**

mat2

-1	8	9
6	**3**	5
5	1	**2**

After the call `int sum = MatrixOp.diagonalOp(mat1, mat2), sum` would contain `21`, as illustrated below.

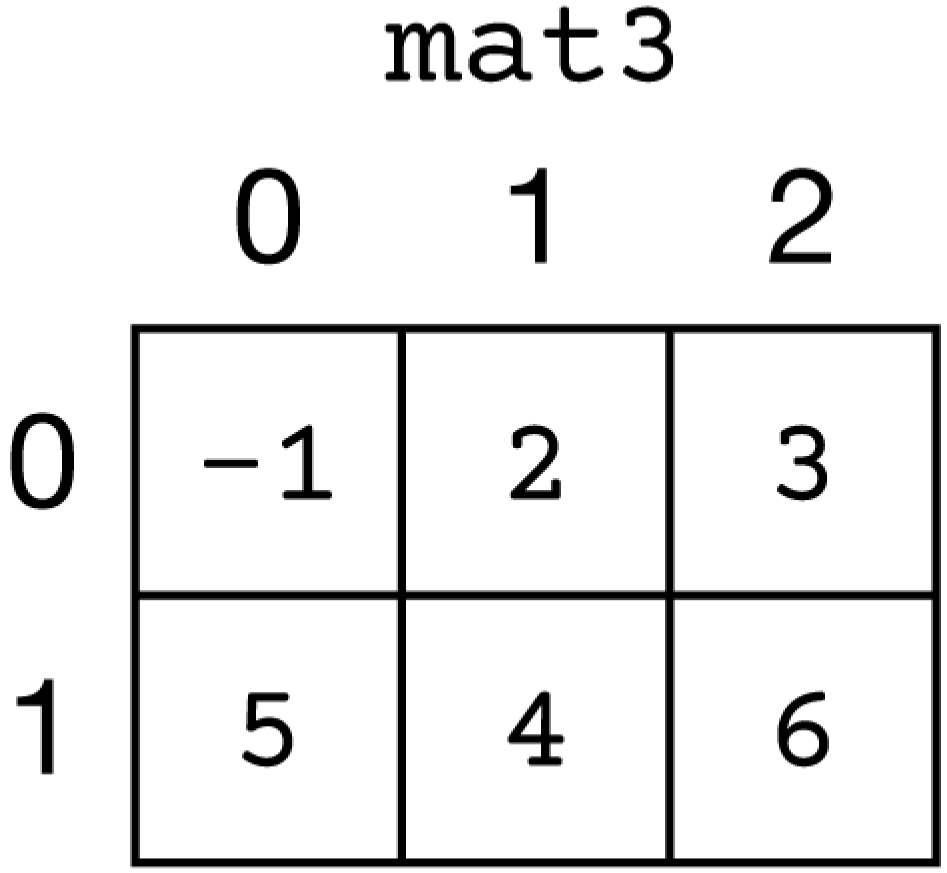
```
sum = (2 * -1) + (5 * 3) + (4 * 2) = 21
```

Complete method `diagonalOp`.

```
/** Returns an integer, as described in part (a).
 * Precondition: matA and matB are 2D arrays that are both square,
 * have at least one row, and have the same dimensions.
 */
public static int diagonalOp(int[][] matA, int[][] matB)
```

(b) Write the method `expandMatrix`, which returns an expanded version of a given 2D array. To expand a 2D array, a new 2D array must be created and filled with values such that each element of the original 2D array occurs a total of four times in the new 2D array, arranged as shown in the example below.

For example, assume that `mat3` is a properly defined 2D array containing the values shown below.



mat3

	0	1	2
0	-1	2	3
1	5	4	6

After the call `int[][] mat4 = MatrixOp.expandMatrix(mat3)`, the array `mat4` would contain the values shown below.

`mat4`

	0	1	2	3	4	5
0	-1	-1	2	2	3	3
1	-1	-1	2	2	3	3
2	5	5	4	4	6	6
3	5	5	4	4	6	6

Complete method `expandMatrix`.

```
/** Returns a 2D array, as described in part (b).
 * Precondition: matA is a 2D array with at least one row and
 * at least one column.
 */
public static int[][] expandMatrix(int[][] matA)
```