4. A grayscale image is represented by a 2-dimensional rectangular array of pixels (picture elements). A pixel is an integer value that represents a shade of gray. In this question, pixel values can be in the range from 0 through 255, inclusive. A black pixel is represented by 0, and a white pixel is represented by 255.

   The declaration of the `GrayImage` class is shown below. You will write two unrelated methods of the `GrayImage` class.

```
public class GrayImage
{
  public static final int BLACK = 0;
  public static final int WHITE = 255;


  /** The 2-dimensional representation of this image. Guaranteed not to be null.
   *  All values in the array are within the range [BLACK, WHITE], inclusive.
   */
  private int[][] pixelValues;


  /** @return the total number of white pixels in this image.
   *  Postcondition: this image has not been changed.
   */
  public int countWhitePixels()
  {  /* to be implemented in part (a) */  }


  /** Processes this image in row-major order and decreases the value of each pixel at
   *  position (row, col) by the value of the pixel at position (row + 2, col + 2) if it exists.
   *  Resulting values that would be less than BLACK are replaced by BLACK.
   *  Pixels for which there is no pixel at position (row + 2, col + 2) are unchanged.
   */
  public void processImage()
  {  /* to be implemented in part (b) */  }
}
```

   (a) Write the method `countWhitePixels` that returns the number of pixels in the image that contain the value `WHITE`. For example, assume that `pixelValues` contains the following image.

	0	1	2	3	4
0	**255**	184	178	84	129
1	84	**255**	**255**	130	84
2	78	**255**	0	0	78
3	84	130	**255**	130	84

   A call to `countWhitePixels` method would return 5 because there are 5 entries (shown in boldface) that have the value `WHITE`.

Complete method `countWhitePixels` below.

```
/** @return the total number of white pixels in this image.
 *  Postcondition: this image has not been changed.
 */
public int countWhitePixels()
```

Part (b) begins on page 20.

(b) Write the method `processImage` that modifies the image by changing the values in the instance variable `pixelValues` according to the following description. The pixels in the image are processed one at a time in row-major order. Row-major order processes the first row in the array from left to right and then processes the second row from left to right, continuing until all rows are processed from left to right. The first index of `pixelValues` represents the row number, and the second index represents the column number.

The pixel value at position (row, col) is decreased by the value at position (row + 2, col + 2) if such a position exists. If the result of the subtraction is less than the value `BLACK`, the pixel is assigned the value of `BLACK`. The values of the pixels for which there is no pixel at position (row + 2, col + 2) remain unchanged. You may assume that all the original values in the array are within the range [`BLACK`, `WHITE`], inclusive.

The following diagram shows the contents of the instance variable `pixelValues` before and after a call to `processImage`. The values shown in boldface represent the pixels that could be modified in a grayscale image with 4 rows and 5 columns.

Before Call to `processImage`

	0	1	2	3	4
0	**221**	**184**	**178**	84	135
1	**84**	**255**	**255**	130	84
2	78	255	0	0	78
3	84	130	255	130	84

After Call to `processImage`

	0	1	2	3	4
0	**221**	**184**	**100**	84	135
1	**0**	**125**	**171**	130	84
2	78	255	0	0	78
3	84	130	255	130	84

Information repeated from the beginning of the question

```
public class GrayImage

public static final int BLACK = 0
public static final int WHITE = 255
private int[][] pixelValues
public int countWhitePixels()
public void processImage()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2012 SCORING GUIDELINES

Question 4: GrayImage

Part (a)	`countWhitePixels`	**4 points**

Intent: *Return the number of white pixels in the image*

+1 Accesses all entries in `pixelValues` (*no bounds errors*)

+1 Compares an entry of array with `WHITE` or with 255 in context of iteration

+1 Initializes and increments a counter

+1 Returns correct count of number of white pixels in `pixelValues`

Part (b)	`processImage`	**5 points**

Intent: *Process elements of* `pixelValues` *and apply specified formula*

+1 Accesses all necessary elements in at least one row or one column of `pixelValues`

+1 Accesses all necessary elements of `pixelValues` (*no bounds errors*)

+1 Decrements element at index `[a][b]` by the original value found in element at index `[a+2][b+2]`

+1 Modifies all and only appropriate elements of `pixelValues` (*changes must not affect last two rows and columns*)

+1 Assigns `BLACK` or 0 to elements of `pixelValues` that would otherwise have a value less than `BLACK` (negative value)

Question-Specific Penalties

-1 (z) Attempts to return a value from `processImage`

AP® COMPUTER SCIENCE A
2012 CANONICAL SOLUTIONS

Question 4: GrayImage

Part (a):

```
public int countWhitePixels() {
   int whitePixelCount = 0;
   for (int[] row : this.pixelValues) {
      for (int pv : row) {
         if (pv == this.WHITE) {
            whitePixelCount++;
         }
      }
   }
   return whitePixelCount;
}
```

Part (a): Alternative solution

```
public int countWhitePixels() {
   int whitePixelCount = 0;
   for (int row = 0; row < pixelValues.length; row++) {
      for (int col = 0; col < pixelValues[0].length; col++) {
         if (pixelValues[row][col] == WHITE) {
            whitePixelCount++;
         }
      }
   }
   return whitePixelCount;
}
```

Part (b):

```
public void processImage() {
   for (int row = 0; row < this.pixelValues.length-2; row++) {
      for (int col = 0; col < this.pixelValues[0].length-2; col++) {
         this.pixelValues[row][col] -= this.pixelValues[row+2][col+2];
         if (this.pixelValues[row][col] < BLACK) {
            this.pixelValues[row][col] = BLACK;
         }
      }
   }
}
```

Part (b): Alternative solution

```
public void processImage() {
   for (int row = 0; row < this.pixelValues.length; row++) {
      for (int col = 0; col < this.pixelValues[0].length; col++) {
         if (row + 2 < pixelValues.length &&
               col + 2 < pixelValues[row].length) {
            this.pixelValues[row][col] -= this.pixelValues[row+2][col+2];
            if (this.pixelValues[row][col] < BLACK) {
               this.pixelValues[row][col] = BLACK;
            }
         }
      }
   }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.