Complete method `toBeLabeled` below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

Part (b) begins on page 16.

GO ON TO THE NEXT PAGE.

(b) Write the `Crossword` constructor. The constructor should initialize the crossword puzzle grid to have the same dimensions as the parameter `blackSquares`. Each element of the puzzle grid should be initialized with a reference to a `Square` object with the appropriate color and number. The number is positive if the square is labeled and 0 if the square is not labeled.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

GO ON TO THE NEXT PAGE.

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

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
```

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width.* Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

Total number of letters in words: 14
Number of gaps between words: 3
Basic gap width: 2
Leftover spaces: 0

Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	P			C	O	M	P			S	C	I			R	O	C	K	S

Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

Total number of letters in words: 15
Number of gaps between words: 3
Basic gap width: 1
Leftover spaces: 2

The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	R	E	E	N			E	G	G	S			A	N	D		H	A	M

Example 3: `wordList: ["BEACH", "BALL"]`

Total number of letters in words: 9
Number of gaps between words: 1
Basic gap width: 11
Leftover spaces: 0

Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	E	A	C	H												B	A	L	L

You will implement three `static` methods in a class named `StringFormatter` that is not shown.

AP® COMPUTER SCIENCE A
2016 CANONICAL SOLUTIONS

Question 4: String Formatter

Part (a):

```
public static int totalLetters(List<String> wordList)
{
      int total = 0;

      for (String word : wordList)
      {
            total += word.length();
      }
      return total;
}
```

Part (b):

```
public static int basicGapWidth(List<String> wordList, int formattedLen)
{
      return (formattedLen - totalLetters(wordList)) / (wordList.size()-1);
}
```

Part (c):

```
public static String format(List<String> wordList, int formattedLen)
{
      String formatted = "";
      int gapWidth = basicGapWidth(wordList, formattedLen);
      int leftovers = leftoverSpaces(wordList, formattedLen);

      for (int w = 0; w < wordList.size() - 1; w++)
      {
            formatted = formatted + wordList.get(w);
            for (int i = 0; i < gapWidth; i++)
            {
                  formatted = formatted + " ";
            }
            if (leftovers > 0)
            {
                  formatted = formatted + " ";
                  leftovers--;
            }
      }
      formatted = formatted + wordList.get(wordList.size() - 1);

      return formatted;
}
```