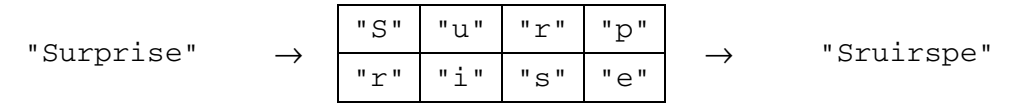
4. In this question you will write two methods for a class `RouteCipher` that encrypts (puts into a coded form) a message by changing the order of the characters in the message. The route cipher fills a two-dimensional array with single-character substrings of the original message in row-major order, encrypting the message by retrieving the single-character substrings in column-major order.

   For example, the word `"Surprise"` can be encrypted using a 2-row, 4-column array as follows.



   An incomplete implementation of the `RouteCipher` class is shown below.

```
public class RouteCipher
{
  /** A two-dimensional array of single-character strings, instantiated in the constructor */
  private String[][] letterBlock;

  /** The number of rows of letterBlock, set by the constructor */
  private int numRows;

  /** The number of columns of letterBlock, set by the constructor */
  private int numCols;

  /** Places a string into letterBlock in row-major order.
   *  @param str the string to be processed
   *  Postcondition:
   *    if str.length() < numRows * numCols, "A" is placed in each unfilled cell
   *    if str.length() > numRows * numCols, trailing characters are ignored
   */
  private void fillBlock(String str)
  {  /* to be implemented in part (a) */  }

  /** Extracts encrypted string from letterBlock in column-major order.
   *  Precondition: letterBlock has been filled
   *  @return the encrypted string from letterBlock
   */
  private String encryptBlock()
  {  /* implementation not shown */  }

  /** Encrypts a message.
   *  @param message the string to be encrypted
   *  @return the encrypted message;
   *          if message is the empty string, returns the empty string
   */
  public String encryptMessage(String message)
  {  /* to be implemented in part (b) */  }

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

(a) Write the method `fillBlock` that fills the two-dimensional array `letterBlock` with one-character strings from the string passed as parameter `str`.

The array must be filled in row-major order—the first row is filled from left to right, then the second row is filled from left to right, and so on, until all rows are filled.

If the length of the parameter `str` is smaller than the number of elements of the array, the string `"A"` is placed in each of the unfilled cells. If the length of `str` is larger than the number of elements in the array, the trailing characters are ignored.

For example, if `letterBlock` has 3 rows and 5 columns and `str` is the string `"Meet at noon"`, the resulting contents of `letterBlock` would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"n"	"o"
"o"	"n"	"A"	"A"	"A"

If `letterBlock` has 3 rows and 5 columns and `str` is the string `"Meet at midnight"`, the resulting contents of `letterBlock` would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"m"	"i"
"d"	"n"	"i"	"g"	"h"

The following expression may be used to obtain a single-character string at position `k` of the string `str`.

```
str.substring(k, k + 1)
```

Complete method `fillBlock` below.

```
/** Places a string into letterBlock in row-major order.
 *  @param str the string to be processed
 *  Postcondition:
 *    if str.length() < numRows * numCols, "A" is placed in each unfilled cell
 *    if str.length() > numRows * numCols, trailing characters are ignored
 */
private void fillBlock(String str)
```

AP® COMPUTER SCIENCE A
2011 SCORING GUIDELINES

Question 4: Cipher

Part (a) `fillBlock` **3½ points**

Intent: *Fill* `letterBlock` *in row-major order from parameter; pad block or truncate string as needed*

- **+½** Copies at least one substring from parameter to `letterBlock`
- **+½** Completely fills `letterBlock` from parameter if possible (*no bounds errors in* `letterBlock` *or parameter*)
- **+1** Results in a distribution of all consecutive one-character substrings from parameter to `letterBlock` (*ignores surplus characters*)
- **+½** Copies these one-character substrings from parameter to `letterBlock` in such a way that the result is in row-major order
- **+1** Pads `letterBlock` with `"A"` if and only if parameter is shorter than block size

Part (b) `encryptMessage` **5½ points**

Intent: *Return encrypted string created by repeatedly invoking* `fillBlock` *and* `encryptBlock` *on substrings of parameter and concatenating the results*

- **+2** Partition parameter
  - **+½** Returns the empty string if the parameter is the empty string
  - **+½** Creates substrings of parameter that progress through the parameter string (*can overlap or skip*)
  - **+1** Processes every character in parameter exactly once (*no bounds errors*)
- **+3** Fill and encrypt a block, concatenate results
  - **+½** Invokes `fillBlock` with parameter or substring of parameter
  - **+½** Invokes `fillBlock` on more than one substring of parameter
  - **+½** Invokes `encryptBlock` after each invocation of `fillBlock`
  - **+½** Concatenates encrypted substrings of parameter
  - **+1** Builds complete, encrypted message
- **+½** Return resulting built string

Question-Specific Penalties

- **–1½** Use of identifier with no apparent resemblance to `letterBlock` for two-dimensional array

AP® COMPUTER SCIENCE A
2011 CANONICAL SOLUTIONS

Question 4: Cipher

Part (a):

```
private void fillBlock(String str) {
  int pos = 0;
  for (int r = 0; r < this.numRows; r++ ) {
    for (int c = 0; c < this.numCols; c++ ) {
      if (pos < str.length()) {
        this.letterBlock[r][c] = str.substring(pos, pos+1);
        pos++;
      } else {
        this.letterBlock[r][c] = "A";
      }
    }
  }
}
```

// Alternative solution

```
private void fillBlock(String str) {
  for (int r = 0; r < this.numRows; r++ ) {
    for (int c = 0; c < this.numCols; c++ ){
      if (str.length() > (c + (r * this.numCols))) {
        this.letterBlock[r][c] = str.substring(c + r * this.numCols,
                                               1 + c + r * this.numCols);
      } else {
        this.letterBlock[r][c] = "A";
      }
    }
  }
}
```

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.

AP® COMPUTER SCIENCE A
2011 CANONICAL SOLUTIONS

Question 4: Cipher (continued)

Part (b):

```
public String encryptMessage(String message) {
  String encryptedMessage = "";
  int chunkSize = this.numRows * this.numCols;
  while (message.length() > 0) {
    if (chunkSize > message.length()) {
      chunkSize = message.length();
    }
    fillBlock(message);
    encryptedMessage += encryptBlock();
    message = message.substring(chunkSize);
  }
  return encryptedMessage;
}
```

// Alternative solution

```
public String encryptMessage(String message) {
  if (message.length() == 0) return "";
  fillBlock(message);
  if (message.length() <= this.numRows * this.numCols) {
    return encryptBlock();
  }
  return (encryptBlock() +
          encryptMessage(message.substring(this.numRows * this.numCols)));
}
```