AP® COMPUTER SCIENCE A
2011 GENERAL SCORING GUIDELINES

Apply the question-specific rubric first; the question-specific rubric *always* takes precedence.
Penalties: The penalty categorization below is for cases not covered by the question-specific rubric. Points can only be deducted in a part of the question that has earned credit via the question-specific rubric, and no section may have a negative point total. A given penalty can be assessed **only once** in a question, even if it occurs on different parts of that question. A maximum of 3 penalty points may be assessed over the entire question.

Nonpenalized Errors

spelling/case discrepancies if no ambiguity*

local variable not declared if other variables are declared in some part

use of keyword as identifier

`[]` vs. `()` vs. `<>`

`=` instead of `==` (and vice versa)

`length`/`size` confusion for array, `String`, and `ArrayList`, with or without `()`

`private` qualifier on local variable

extraneous code with no side effect; *e.g., precondition check*

common mathematical symbols for operators (x • ÷ ≤ ≥ < > ≠)

missing { } where indentation clearly conveys intent and { } used elsewhere

default constructor called without parens;
e.g., `new Critter;`

missing `( )` on parameter-less method call

missing `( )` around `if/while` conditions

missing `;` when majority are present

missing `public` on class or constructor header

extraneous `[]` when referencing entire array

`[i,j]` instead of `[i][j]`

extraneous size in array declaration,
e.g., `int[size] nums = new int[size];`

Minor Errors (½ point)

confused identifier *(e.g.,* `len` *for* `length` *or* `left()` *for* `getLeft()` *)*

local variables used but none declared

missing `new` in constructor call

modifying a constant (`final`)

use of `equals` or `compareTo` method on primitives, *e.g.,* `int x; …x.equals(val)`

array/collection access confusion (`[]` `get`)

assignment dyslexia,
e.g., `x + 3 = y;` *for* `y = x + 3;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, *e.g.,* `a = method(int x)`

missing `public` from method header when required

`"false"`/`"true"` or 0/1 for boolean values

`"null"` for `null`

Applying Minor Penalties (½ point):
A minor infraction that occurs **exactly once** when the same concept is **correct two or more times** is regarded as an oversight and **not penalized**. A minor penalty **must be assessed** if the item is the **only instance, one of two**, or occurs **two or more times**.

Major Errors (1 point)

extraneous code that causes side effect; *e.g., information written to output*

interface or class name instead of variable identifier; *e.g.,* `Bug.move()` *instead of* `aBug.move()`

`aMethod(obj)` instead of `obj.aMethod()`

attempt to use private data or method when not accessible

destruction of persistent data *(e.g., changing value referenced by parameter)*

use of class name in place of `super` in constructor or method call

`void` method (or constructor) returns a value

* *Spelling and case discrepancies for identifiers fall under the "nonpenalized" category only if the correction can be* ***unambiguously*** *inferred from context; for example, "*`ArayList`*" instead of "*`ArrayList`*". As a counterexample, note that if a student declares "*`Bug bug;`*" then uses "*`Bug.move()`*" instead of "*`bug.move()`*", the context does* ***not*** *allow for the reader to assume the object instead of the class.*

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";
      }
    }
  }
}
```

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)));
}
```

4A,

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)
{
    int k = 0;
    for (int row = 0; row < numRows; row++)
    {
        for (int col = 0; col < numCols; col++)
        {
            if ( k != str.length)
            {
                letterBlock[row][col] = str.substring(k, k+1);
                k++;
            }
            else
            {
                letterBlock[row][col] = "A";
            }
        }
    }
}
```

Part (b) begins on page 22.

4A2

Complete method encryptMessage below.

```
/** 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)
{
    String encryptedMessage = "";
    int start = 0;
    int end = numRows * numCols;

    while (start < message.length())
    {
        if (start + end > message.length())
        {
            fillBlock(message.substring(start));
            encryptedMessage = encryptedMessage + encryptBlock();
        }
        else
        {
            fillBlock(message.substring(start, start + end));
            encryptedMessage = encryptedMessage + encryptBlock();
            start = start + end;
        }
    }

    return encryptedMessage;
}
```

4B.

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)
{
    int len = str.length();
    int slots = numRows * numCols;

    for ( int i =0; i < slots; i ++ )
    {
       for (int j =0; j < numRows; j ++)
       {
          for (int k=0; k < numCols; k++)
          {
             if ( i < len)
                letterBlock[ j][k] = str.substring(i, i+1);
             else
                letterBlock [j] [k] = "A";
          }
       }
    }
}
```

Part (b) begins on page 22.

GO ON TO THE NEXT PAGE.

4B2

Complete method encryptMessage below.

```
/** 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)
{
    int len = message.length();
    int size = numRows * numCols;
    int NumArrays = (len / size)
        if ( len % size != 0)
            numArrays += 1;
    String mixed = "";
    for (int i = 0; i < numArrays; i++)
    {
        FillBlock( message.subString(i*size, (i+1)*size) );
        mixed = mixed + encryptBlock();
    }

    return mixed;
}
```


GO ON TO THE NEXT PAGE.

4C.

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)
{
    int i = 0;
    // input loop
    for (int j = 0; j < letterBlock.length; j++)
    {
        for (int k = 0; k < letterBlock[j].length; k++)
        {
            letterBlock[j][k] = str.substring(i, i+1);
            i++;
        }
    }

    // fills remaining spots with "A"
    for (int j = 0; j < letterBlock.length; j++)
    {
        for (int k=0; k < letterBlock[j].length; k++)
        {
            if (letterBlock[j][k] == null)
            {
                letterBlock[j][k] = "A";
            }
        }
    }
}
```

Part (b) begins on page 22.

GO ON TO THE NEXT PAGE.

4C2

Complete method encryptMessage below.

```
/** 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)
{
    fillBlock(message);
    String newMessage = "";
    newMessage = newMessage + encryptBlock();
    int rand = Math.random() * 5; // repeats from 0 to 5 times
    for(int i = 0; i < rand; i++)
    {
        encryptMessage(message); // restarts method
    }
}
```


GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2011 SCORING COMMENTARY

Question 4

Overview

This question focused on using two-dimensional arrays and utilizing abstractions. The question required students to manipulate Strings and to demonstrate understanding of how class fields interact. Students were provided with a partial class definition for the `RouteCipher` class, which indicated three relevant fields (a 2D array and the two dimensions of the array) and three methods, two of which were to be written by the student (`fillBlock` and `encryptMessage`). The third method, `encryptBlock`, was understood to be already implemented as per the given documentation and ready to be utilized.

Part (a) asked students to write the method `fillBlock`, whose task was to break apart a String into a sequence of one-character substrings and put them into a 2D array in row-major order. Nested for-loops and use of the `substring` method (of the String class) were the most common ways this was achieved.

Complications arose when the number of characters in the String did not match the size of the 2D `letterBlock`. Per the instructions, if there were extra characters in the String, they were to be ignored. If there were not enough characters in the String, the rest of the spots in the `letterBlock` were to be filled with the String `"A"` — a process called "padding."

Part (b) asked students to write the method `encryptMessage`, whose task was to take a message String and encrypt it via the `fillBlock` method written in part (a) and the `encryptBlock` method that was given to them. This required breaking the String into appropriately sized chunks that could be sent to `fillBlock`, encrypted by `encryptBlock` (which returned the encrypted message), and then combined into one String that contained the entire message in encrypted form.

Sample: 4A
Score: 8

In part (a) the student correctly implements the `fillBlock` method. If the length of the parameter `str` is less than or equal to the number of cells in `letterBlock`, all consecutive one-character substrings of `str` are correctly copied into the `letterBlock` array. The copying is done in row-major order. All unfilled cells in the `letterBlock` array are filled with `"A"`. If the length of `str` is greater than the number of cells in `letterBlock`, the excess one-character substrings of `str` are ignored. Part (a) earned all 3½ points.

In part (b) the student does not implement the `encryptMessage` method correctly. If the parameter `message` is the empty string, the empty string is returned. Otherwise, `message` is partitioned into disjoint substrings. The number of partitions depends on the length of `message`. If the length of the last partition is less than `numRows*numCols`, then the while loop variable is not incremented, which results in an infinite loop. Ignoring this error, each partition is passed to `fillBlock`, and then `encryptBlock` is called to encrypt the partition. All encrypted partitions are concatenated in order, and the result of the concatenation is returned. Part (b) earned 4½ points.

Question 4 (continued)

Sample: 4B
Score: 6

In part (a) the student does not implement the `fillBlock` method correctly. One-character substrings of the parameter `str` are copied into all cells of the `letterBlock` array in row-major order. If the length of `str` is greater than or equal to `numRows*numCols`, then the inside loop causes only the one-character substring calculated last to be copied into all cells of `letterBlock`. If the length of `str` is less than `numRows*numCols`, then `"A"` is copied into all cells of `letterBlock`. Part (a) earned 1½ points.

In part (b) the student does not implement the `encryptMessage` method correctly. If the parameter `message` is the empty string, then the empty string is returned, because the loop does not execute. Otherwise `message` is partitioned into disjoint substrings. The number of partitions depends on the length of `message`. If the length of final partition of `message` is less than `numRows*numCols`, then a bounds error occurs in the substring call. Ignoring this error, each partition is passed to `fillBlock`, and then `encryptBlock` is called to encrypt the partition. All encrypted partitions are concatenated in order, and the result of the concatenation is returned. Part (b) earned 4½ points.

Sample: 4C
Score: 3

In part (a) the student does not implement the method `fillBlock` correctly. If the variable `i` is greater than or equal to the length of `str`, then a bounds error occurs in the substring call. Ignoring this error, if the length of the parameter `str` is less than or equal to the number of cells in `letterBlock`, all consecutive one-character substrings of `str` are correctly copied into the `letterBlock` array. The copying is done in row-major order. Because `null` cannot be assumed to be values in the `letterBlock` array padding, `letterBlock` is not done correctly. Part (a) earned 2 points.

In part (b) the student does not implement the method `encryptMessage` correctly. If the parameter `message` is the empty string, nothing is returned. No partitioning of `message` occurs. The `fillBlock` method is called correctly in each of the recursive calls to `encryptMessage` but always with the same parameter. The method `encryptBlock` is called correctly after each call to `fillBlock`, but the concatenation variable contains only the last encryption, because it is initialized before each concatenation operation. There is no return statement. Part (b) earned 1 point.