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 1: Sound

Part (a)	`limitAmplitude`	**4½ points**

Intent: Change elements of `samples` *that exceed* `±limit`; *return number of changes made*

+3 Identify elements of `samples` to be modified and modify as required
- **+1** Consider elements of `samples`
  - **+½** Accesses more than one element of `samples`
  - **+½** Accesses every element of `samples` (*no bounds errors*)
- **+2** Identify and change elements of `samples`
  - **+½** Compares an element of `samples` with `limit`
  - **+½** Changes at least one element to `limit` or `-limit`
  - **+1** Changes all and only elements that exceed `±limit` to `limit` or `-limit` appropriately

+1½ Calculate and return number of changed elements of `samples`
- **+1** Initializes and updates a counter to achieve correct number of changed samples
- **+½** Returns value of an updated counter (*requires array access*)

Part (b)	`trimSilenceFromBeginning`	**4½ points**

Intent: Remove leading elements of `samples` *that have value of* 0, *potentially resulting in array of different length*

+1½ Identify leading-zero-valued elements of `samples`
- **+½** Accesses every leading-zero element of `samples`
- **+½** Compares 0 and an element of `samples`
- **+½** Compares 0 and multiple elements of `samples`

+1 Create array of proper length
- **+½** Determines correct number of elements to be in resulting array
- **+½** Creates new array of determined length

+2 Remove silence values from `samples`
- **+½** Copies some values other than leading-zero values
- **+1** Copies all and only values other than leading-zero values, preserving original order
- **+½** Modifies instance variable `samples` to reference newly created array

Question-Specific Penalties

–1 Array identifier confusion (e.g., `value` instead of `samples`)
–½ Array/collection modifier confusion (e.g., using `set`)

AP® COMPUTER SCIENCE A
2011 CANONICAL SOLUTIONS

Question 1: Sound

Part (a):

```
public int limitAmplitude(int limit) {
  int numChanged = 0;
  for (int i = 0; i < this.samples.length; i++) {
    if (this.samples[i] < -limit) {
      this.samples[i] = -limit;
      numChanged++;
    }
    if (this.samples[i] > limit) {
      this.samples[i] = limit;
      numChanged++;
    }
  }
  return numChanged;
}
```

Part (b):

```
public void trimSilenceFromBeginning() {
  int i = 0;
  while (this.samples[i] == 0) {
    i++;
  }
  int[] newSamples = new int[this.samples.length - i];
  for (int j = 0; j < newSamples.length; j++) {
    newSamples[j] = this.samples[j+i];
  }
  this.samples = newSamples;
}
```

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.
 *  Values greater than limit are changed to limit.
 *  Values less than -limit are changed to -limit.
 *  @param limit the amplitude limit
 *          Precondition: limit ≥ 0
 *  @return the number of values in this sound that this method changed
 */
public int limitAmplitude(int limit)
{
    int count = 0;
    for(int i = 0 ; i < samples.length ; i++)
    {
        if(samples[i] > limit)
        {
            samples[i] = limit;
            count ++;
        }
        else if (samples[i] < -limit)
        {
            samples[i] = -limit;
            count ++;
        }
    }
    return count;
}
```

Part (b) begins on page 6.

Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.
 *  Silence is represented by a value of 0.
 *  Precondition: samples contains at least one nonzero value
 *  Postcondition: the length of samples reflects the removal of starting silence
 */
public void trimSilenceFromBeginning()
{
  int nonZero = 0;
  while (samples[nonZero] == 0)
      nonZero++;
  int[] trimmed = new int[samples.length - nonZero];
  for(int i = 0; i < trimmed.length; i++)
       trimmed[i] = samples[nonZero + i];
  samples = trimmed
 }
```

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.
  *  Values greater than limit are changed to limit.
  *  Values less than -limit are changed to -limit.
  *  @param limit the amplitude limit
  *          Precondition: limit ≥ 0
  *  @return the number of values in this sound that this method changed
  */
public int limitAmplitude(int limit) {
    int count = 0;
    for(int k=0; k < sampes.length; k++) {
      if(samples[k] > 2000 || samples[k] < -2000) {
         if(samples[k] < 0 ) {
            samples[k] = -1 * limit;
         }
         else {
            samples[k] = limit;
         }
         count++;
      }
   / }
    return count;
}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.
 *  Silence is represented by a value of 0.
 *  Precondition: samples contains at least one nonzero value
 *  Postcondition: the length of samples reflects the removal of starting silence
 */
public void trimSilenceFromBeginning(){
 int zeroCount = 0;
 if(samples[0] == 0){
     for(int k=0; k < samples.length; k++){
     if(samples[k] == 0){
        zeroCount ++;
       }
       else {
       break;
       }
    }
   for(int x = 0; x < samples.length-zeroCount; x++){
      samples[x] = samples[x + zeroCount];
   }
   for(int y=samples.length-zeroCount; y< samples.length; y++){
      samples[y]= null;
   }
 }
}
```

GO ON TO THE NEXT PAGE.

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.
 *  Values greater than limit are changed to limit.
 *  Values less than -limit are changed to -limit.
 *  @param limit the amplitude limit
 *           Precondition: limit ≥ 0
 *  @return the number of values in this sound that this method changed
 */
public int limitAmplitude(int limit)
```

```
{
        int counter;
        for(int temp : samples)
        {
           if(temp > limit)
           {    temp = limit;
                counter ++;
           }
           else if(temp < (limit * -1))
           {    temp = (limit * -1);
                counter ++
           }
        }
        return counter;
}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

1C2

Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.
 *  Silence is represented by a value of 0.
 *  Precondition: samples contains at least one nonzero value
 *  Postcondition: the length of samples reflects the removal of starting silence
 */
public void trimSilenceFromBeginning()
```

```
{
        for(int x = 0 ; x < samples.length() ; x ++)
        {
            if( samples.get(x) != 0)
            {
                return;
            }
            else
            {   samples.remove(x);
            }
        }
        return;
}
```

Question 1

Overview

This question focused on array traversals, comparing array elements with a specified limit, and deleting elements from an array based on a condition. Students were given a class containing an array, `samples`, which contained sound values, and were asked to write two methods for processing those values.

The first method, `limitAmplitude`, required students to compare each element of the array to a specified limit, which was passed as a parameter. Values greater than `limit` were to be changed to `limit` and values less than `-limit` were changed to `-limit`. The method also required students to count and return the number of those changes made to the array.

The second method, `trimSilenceFromBeginning`, required students to create a new array that contained the same values in the same order as the original array but without the leading zeros. The solution required the instance variable `samples` to be updated to refer to the newly created array.

Sample: 1A
Score: 9

In part (a) the student correctly writes a loop that examines all elements of the array. The loop correctly compares all elements to the limit value passed as a parameter, changes the value in the array when necessary, and counts the number of changes made correctly. Variables are correctly initialized, and the final count is returned. Part (a) earned all 4½ points.

In part (b) the first loop counts the number of leading-zero elements. An array of the determined length is created. The second loop traverses the remaining elements of the array to fill the newly created array. The instance variable `samples` is updated to reference the newly created array. Part (b) earned all 4½ points.

Sample: 1B
Score: 6.5 (rounded to 7)

In part (a) the student correctly writes a loop that examines all elements of the array. The loop incorrectly compares elements to the value 2000 rather than the value passed as a parameter, resulting in no credit for the comparison. The array element is changed to `+/- limit`, but the change is based on a bad constant in the conditional, resulting in no credit for changing all and only elements. The student received credit for initializing, counting, and returning the number of elements that were changed. Part (a) earned 3 points.

In part (b) the first loop counts the number of leading-zero elements. The code computes the number of values after the leading zeros but fails to create a new array. The code copies all the values other than leading-zero values into the instance variable `samples`, but assigning `null` does not reduce the size of the array. Part (b) earned 3½ points.

Question 1 (continued)

Sample: 1C
Score: 2

In part (a) the student uses a for-each loop to traverse the array. This allows the code to access, compare, and count the number of elements that exceed `+/- limit` appropriately but does not correctly change the elements in the array. No credit was received for initializing and updating a counter, because the student does not initialize the counter. Part (a) earned 2 points.

In part (b) the solution earned ½ point for comparing an element of the array to 0 but then was penalized for `get/[]` confusion, as outlined in the General Scoring Guidelines. Because the `remove` method is not defined for arrays, all of the remaining points were lost in this part. If the data structure had been an `ArrayList` instead of an array, the `remove` method would have simplified the solution to the problem significantly, because it does not require array manipulation. Part (b) earned 0 points.