AP® COMPUTER SCIENCE A
2016 GENERAL SCORING GUIDELINES

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., writing to output, failure to compile)

x) Local variables used but none declared

y) Destruction of persistent data (e.g., changing value referenced by parameter)

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context. For example,* "ArayList" *instead of* "ArrayList". *As a counter example, note that if the code 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.*

Question 1: Climbing Club

Part (a) `addClimb` (append) **2 points**

Intent: *Create new* `ClimbInfo` *using data from parameters and append to* `climbList`

+1 Creates new `ClimbInfo` object using parametric data correctly

+1 Appends the created object to `climbList`
(*no bounds error and no destruction of existing data*)
(*point not awarded if inserted more than once*)

Part (b) `addClimb` (alphabetical) **6 points**

Intent: *Create new* `ClimbInfo` *object using data from parameters and insert into* `climbList`, *maintaining alphabetical order*

+1 Creates new `ClimbInfo` object(s), using parametric data correctly

+1 Compares `peakName` value with value retrieved from object in list (*must use* `getName`)

+1 Inserts object into list based on a comparison (other than equality) with object in list
(*point not awarded if inserted more than once*)

+1 Compares parametric data with all appropriate entries in `climbList` (*no bounds error*)

+1 Inserts new `ClimbInfo` object into `climbList` (*no destruction of existing data*)

+1 Inserts new `ClimbInfo` object into `climbList` once and only once in maintaining alphabetical order (*no destruction of existing data*)

Part (c) analysis **1 point**

Intent: *Analyze behavioral differences between* **append** *and* **alphabetical** *versions of* `addClimb`

+1 (i) NO (ii) YES Both must be answered correctly

Question-Specific Penalties

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

Question 1: Climbing Club

Part (a):

```
public void addClimb(String peakName, int climbTime) {
   this.climbList.add(new ClimbInfo(peakName, climbTime));
}
```

Part (b):

```
public void addClimb(String peakName, int climbTime) {
   for (int i = 0; i < this.climbList.size(); i++) {
      if (peakName.compareTo(this.climbList.get(i).getName()) <= 0) {
         this.climbList.add(i, new ClimbInfo(peakName, climbTime));
         return;
      }
   }
   this.climbList.add(new ClimbInfo(peakName, climbTime));
}
```

Part (c):

NO

YES

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.

Question 2: TokenPass

Part (a)	`TokenPass` constructor	**4 points**

Intent: *Create* `TokenPass` *object and correctly initialize game state*

+1 Creates instance variable `board` as `int` array of size `playerCount`

+1 Computes a random number between 1 and 10, inclusive, and a random number between 0 and `playerCount-1`, inclusive

+1 Initializes all entries in `board` with computed random value *(no bounds errors)*

+1 Initializes instance variable `currentPlayer` to computed random value

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

Intent: *Distribute all tokens from* `currentPlayer` *position to subsequent positions in array*

+1 Uses initial value of `board[currentPlayer]` to control distribution of tokens

+1 Increases at least one `board` entry in the context of a loop

+1 Starts distribution of tokens at correct board entry

+1 Distributes next token (if any remain) to position 0 after distributing to highest position in board

+1 On exit: token count at each position in `board` is correct

Question-Specific Penalties

-2 (v) Consistently uses incorrect array name instead of `board`

-1 (y) Destruction of persistent data (`currentPlayer`)

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

Question 2: TokenPass

Part (a):

```
public TokenPass(int playerCount)
{
    board = new int[playerCount];
    for (int i = 0; i < playerCount; i++){
        board[i] = 1 + (int) (10 * Math.random());
    }
    currentPlayer = (int) (playerCount * Math.random());
}
```

Part (b):

```
public void distributeCurrentPlayerTokens()
{
    int nextPlayer = currentPlayer;
    int numToDistribute = board[currentPlayer];
    board[currentPlayer] = 0;

    while (numToDistribute > 0){
        nextPlayer = (nextPlayer + 1) % board.length;
        board[nextPlayer]++;
        numToDistribute--;
    }
}
```

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.

Question 3: Seating Chart

Part (a) `SeatingChart` constructor **5 points**

Intent: *Create* `SeatingChart` *object from list of students*

+1 `seats = new Student[rows][cols];` (or equivalent code)

+1 Accesses all elements of `studentList` *(no bounds errors on* `studentList`*)*

+1 Accesses all necessary elements of seats array *(no bounds errors on seats array, point lost if access not column-major order)*

+1 Assigns value from `studentList` to at least one element in seats array

+1 On exit: All elements of `seats` have correct values (*minor loop bounds errors ok*)

Part (b) `removeAbsentStudents` **4 points**

Intent: *Remove students with more than given number of absences from seating chart and return count of students removed*

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

+1 Calls `getAbsenceCount()` on `Student` object (*point lost if null case not handled correctly)*

+1 Assigns `null` to all elements in seats array when absence count for occupying student > `allowedAbsences` *(point lost if seats array element changed in other cases)*

+1 Computes and returns correct number of students removed

Question-Specific Penalties

-2 (v) Consistently uses incorrect array name instead of `seats` or `studentList`

Question 3: SeatingChart

Part (a):

```
public SeatingChart(List<Student> studentList, int rows, int cols){
   seats=new Student[rows][cols];
   int studentIndex=0;
   for (int col = 0; col < cols; col++){
      for (int row = 0; row < rows; row++){
         if (studentIndex < studentList.size()){
            seats[row][col] = studentList.get(studentIndex);
            studentIndex++;
         }
      }
   }
}
```

Part (a) alternate:

```
public SeatingChart(List<Student> studentList, int rows, int cols){
   seats=new Student[rows][cols];
   int row=0;
   int col=0;
   for (Student student : studentList){
      seats[row][col]=student;
      row++;
      if (row==rows){
         row=0;
         col++;
      }
   }
}
```

Part (b):

```
public int removeAbsentStudents(int allowedAbsences){
   int count = 0;
   for (int row=0; row < seats.length; row++){
      for (int col=0; col < seats[0].length; col++){
         if (seats[row][col] != null &&
               seats[row][col].getAbsenceCount() > allowedAbsences){
            seats[row][col]=null;
            count++;
         }
      }
   }
   return count;
}
```

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.

Question 4: Number Group

Part (a)	Interface: **`NumberGroup`**	2 points

Intent: *Define interface to represent a number group*

- **+1** `interface NumberGroup` (*point lost if visibility private*)
- **+1** `boolean contains(int num);`
(*point lost if visibility not public or extraneous code present*)

Part (b)	Class: **`Range`**	5 points

Intent: *Define implementation of* `NumberGroup` *representing a range of numbers*

- **+1** `class Range implements NumberGroup` (*point lost if visibility private*)
- **+1** Declares appropriate `private` instance variable(s)
- **+1** Uses correct constructor header
- **+1** Initializes instance variables within constructor using parameters
(*point lost if bounds errors occur in container use*)
- **+1** Computes and returns correct value from `contains`
(*point lost for incorrect method header*)

Part (c)	`contains`	2 points

Intent: *Determine whether integer is part of any of the member number groups*

- **+1** Calls `contains` on elements of `groupList` in context of loop *(no bounds errors)*
- **+1** Computes and returns correct value

Question-Specific Penalties

- **-1** (s) Inappropriate use of `static`

Question 4: Number Group

Part (a):

```
public interface NumberGroup
{
    boolean contains(int num);
}
```

Part (b):

```
public class Range implements NumberGroup
{
    private int min;
    private int max;

    public Range(int min, int max)
    {
        this.min=min;
        this.max=max;
    }

    public boolean contains(int num){
        return num >= min && num <= max;
    }
}
```

Part (c):

```
public boolean contains(int num){
       for (NumberGroup group : groupList){
           if (group.contains(num)){
               return true;
           }
       }
       return false;
}
```

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.