AP® Computer Science A
2013 Scoring Guidelines
Revised April 2014

The College Board

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of over 6,000 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT® and the Advanced Placement Program®. The organization also serves the education community through research and advocacy on behalf of students, educators, and schools. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Visit the College Board on the Web: www.collegeboard.org.
AP Central is the official online home for the AP Program: apcentral.collegeboard.org.

AP® COMPUTER SCIENCE A
2013 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.

1-Point Penalty

(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
- Array/collection access confusion (`[] get`)
- `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 `;` provided majority are present and indentation clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- 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, "`ArrayList`" instead of "`ArrayList`". As a counterexample, 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.*

AP® COMPUTER SCIENCE A
2013 SCORING GUIDELINES

Question 1: SongList

Part (a)	`getDownloadInfo`	**4 points**

Intent: *Search download list for requested title and return matching* `DownloadInfo` *object if found.*

+1 Accesses all necessary entries in `downloadList` (*no bounds errors*)

+3 Identifies and returns matching entry in `downloadList`, if it exists

- **+1** Calls `getTitle` on `DownloadInfo` object from `downloadList`
- **+1** Checks for equality between title from list object and `title` parameter (*must use String equality check*)
- **+1** Returns reference to matching object if present; `null` if not (*point not awarded for early return*)

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

Intent: *Update* `downloadList` *with information from list of titles*

+1 Accesses all entries in `titles` *(no bounds error for* `titles`*)*

+1 Calls `getDownloadInfo(`*`title`*`)` to determine whether title from `titles` list exists in `downloadList`

+1 Increments the count in matching `DownloadInfo` object if title is in `downloadList`

+1 Constructs new `DownloadInfo` object (with correct information) if title is not in `downloadList` *(point not awarded if incremented at time of construction)*

+1 Adds constructed object to end of `downloadList` if title is not in `downloadList` *(point not awarded if added more than once)*

Question-Specific Penalties

-1 (g) Uses `getLength`/`getSize` for ArrayList size

-2 (v) Consistently uses incorrect array name instead of `downloadList`/`titles`

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

AP® COMPUTER SCIENCE A
2013 CANONICAL SOLUTIONS

Question 1: SongList

Part (a):

```
public DownloadInfo getDownloadInfo(String title) {
      for (DownloadInfo info : downloadList){
            if (info.getTitle().equals(title)){
                return info;
            }
      }
      return null;
}
```

Part (b):

```
public void updateDownloads(List<String> titles) {
    for (String title : titles) {
        DownloadInfo foundInfo = getDownloadInfo(title);
        if (foundInfo == null){
           downloadList.add(new DownloadInfo(title));
        }
        else {
            foundInfo.incrementTimesDownloaded();
        }
    }
}
```

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
2013 SCORING GUIDELINES

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`

AP® COMPUTER SCIENCE A
2013 CANONICAL SOLUTIONS

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.

AP® COMPUTER SCIENCE A
2013 SCORING GUIDELINES

Question 3: JumpingCritter (GridWorld)

Part (a) `getEmptyLocations` **5 points**

Intent: *Create and return* `ArrayList<Location>` *of all empty locations in grid*

+½ Declares and constructs empty `ArrayList<Location>`

+½ Accesses all locations in `grid` *(no bounds errors)*

+2 Identifies empty location in grid in context of loop
- **+1** Creates new location in grid
- **+1** Determines if created location is empty

+1 Includes all and only identified empty locations in constructed arraylist exactly once

+1 Returns the constructed arraylist (*code must have examined grid*)

Part (b) Class: `JumpingCritter` **4 points**

Intent: *Define extension to* `Critter` *class that jumps to randomly selected empty location in its grid*

+½ `class JumpingCritter extends Critter`

+1½ Override `getMoveLocations`
- **+½** `public ArrayList<Location> getMoveLocations()`
- **+½** `GridWorldUtilities.getEmptyLocations(getGrid())`
- **+½** Returns arraylist containing empty locations

+1 Handles `null` location case correctly in `selectMoveLocation`

+1 Handles random location case correctly (must override `getMoveLocations`)

Question-Specific Penalties

-1 (s) Causes inappropriate state change in world (`Grid`, `Actor`, …)

-1 (t) Overrides `act`

AP® COMPUTER SCIENCE A
2013 CANONICAL SOLUTIONS

Question 3: JumpingCritter (GridWorld)

Part (a):

```
public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
{
  ArrayList<Location> locs = new ArrayList<Location>();

  for (int r = 0; r < grid.getNumRows(); r++){
      for (int c = 0; c < grid.getNumCols(); c++){
          Location locToCheck = new Location(r,c);
          if (grid.get(locToCheck) == null){
              locs.add(locToCheck);
          }
      }
  }

  return locs;
}
```

Part (b):

```
public class JumpingCritter extends Critter {

    public ArrayList<Location> getMoveLocations(){
        return GridWorldUtilities.getEmptyLocations(getGrid());
    }

    public Location selectMoveLocation(ArrayList<Location> locs){
        if (locs.size() == 0){
            return null;
        } else {
            Location newLoc = locs.get((int)(Math.random()*locs.size()));
            return newLoc;
        }
    }
}
```

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
2013 SCORING GUIDELINES

Question 4: SkyView

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

Intent: *Construct* `SkyView` *object from 1D array of scan data*

+1 Constructs correctly-sized 2D array of doubles and assigns to instance variable `view`

+1 Initializes at least one element of `view` with value from element of `scanned` (*must be in context of loop*)

+1 Places consecutive values from `scanned` into at least one row of `view` in original order

+1 Places consecutive values from `scanned` into at least one row of `view` in reverse order

+1 On exit: all elements of `view` have correct values (*no bounds errors on* `view` *or* `scanned`)

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

Intent: *Compute and return average of rectangular section of* `view`*, specified by parameters*

+1 Declares and initializes a `double` accumulator

+1 Adds all and only necessary values from `view` to accumulator (*no bounds errors*)

+1 Computes average of specified rectangular section

+1 Returns the computed average (*computation must involve* `view`)

Question-Specific Penalties

-2 (v) Consistently uses incorrect array name instead of `view/scanned`

AP® COMPUTER SCIENCE A
2013 CANONICAL SOLUTIONS

Question 4: SkyView

Part (a):

```
public SkyView(int numRows, int numCols, double[] scanned)
{
  view = new double[numRows][numCols];
  int i = 0;
  for (int row = 0; row < numRows; row++) {
    if (row % 2 == 0) {
      for (int col = 0; col < numCols; col++) {
        view[row][col] = scanned[i];
        i++;
      }
    }
    else {
      for (int col = numCols - 1; col >= 0; col--) {
        view[row][col] = scanned[i];
        i++;
      }
    }
  }
}
```

Part (b):

```
public double getAverage(int startRow, int endRow, int startCol,
                                                   int endCol)
{
    double sum = 0.0;
    int count = 0;
    for (int row = startRow; row <= endRow; row++){
        for (int col = startCol; col <= endCol; col++){
            sum += view[row][col];
            count++;
        }
    }
    return sum / 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.