AP® COMPUTER SCIENCE A
2012 GENERAL SCORING GUIDELINES

Apply the question-specific 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-specific rubric. No part of a question — (a), (b), or (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 different parts of that question.

1-Point Penalty

(w) Extraneous code that causes a side effect or prevents earning points in the rubric (*e.g., information written to output*)

(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 that causes no side effect
- Extraneous code that is unreachable and would not have earned points in rubric
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared, provided that other variables are declared in some part
- `private` qualifier on local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (x • ÷ ≤ ≥ < > ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` (and vice versa)
- Array/collection element access confusion (`[]` vs. `get` for r-values)
- Array/collection element modification confusion (`[]` vs. `set` for l-values)
- `length/size` confusion for array, `String`, and `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 that line breaks and indentation clearly convey intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if/while` conditions
- Use of local variable outside declared scope (must be within same method body)
- Failure to cast object retrieved from nongeneric collection

* *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 counterexample, note that if the code declares "*`Bug bug;`*" and 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 3: Horse Barn

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

Intent: *Return index of space containing horse with specified name*

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

+1 Checks for `null` reference in array and avoids dereferencing it (*in context of loop*)

+1 Checks for name equality between array element and parameter
(must use `String` *equality check*)

+1 Returns correct index, if present; -1 point if not

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

Intent: *Repopulate* `spaces` *such that the order of all non-*`null` *entries is preserved and all* `null` *entries are found contiguously at the largest indices*

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

+1 Identifies and provides different treatment of `null` and non-`null` elements in array

+1 Assigns element in array to a smaller index
(*must have identified source as non-*`null` *or destination as* `null`)

+1 On exit: The number, integrity, and order of all identified non-`null` elements in `spaces` is preserved, and the number of `null` elements is preserved

+1 On exit: All non-`null` elements in `spaces` are in contiguous locations, beginning at index 0 *(no destruction of data*)

Question-Specific Penalties

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

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

AP® COMPUTER SCIENCE A
2012 CANONICAL SOLUTIONS

Question 3: Horse Barn

Part (a):

```
public int findHorseSpace(String name) {
   for (int i = 0; i < this.spaces.length; i++) {
      if (this.spaces[i]!=null && name.equals(this.spaces[i].getName())) {
         return i;
      }
   }
   return -1;
}
```

Part (b):

```
public void consolidate() {
   for (int i = 0; i < this.spaces.length-1; i++) {
      if (this.spaces[i] == null) {
         for (int j = i+1; j < this.spaces.length; j++) {
            if (this.spaces[j] != null) {
               this.spaces[i] = this.spaces[j];
               this.spaces[j] = null;
               j = this.spaces.length;
            }
         }
      }
   }
}
```

Part (b): Alternative solution (auxiliary with array)

```
public void consolidate() {
   Horse[] newSpaces = new Horse[this.spaces.length];
   int nextSpot = 0;
   for (Horse nextHorse : this.spaces) {
      if (nextHorse != null) {
         newSpaces[nextSpot] = nextHorse;
         nextSpot++;
      }
   }
   this.spaces = newSpaces;
}
```

Part (b): Alternative solution (auxiliary with `ArrayList`)

```
public void consolidate() {
   List<Horse> horseList = new ArrayList<Horse>();
   for (Horse h : this.spaces) {
      if (h != null) horseList.add(h);
   }
   for (int i = 0; i < this.spaces.length; i++) {
      this.spaces[i] = null;
   }
   for (int i = 0; i < horseList.size(); i++) {
      this.spaces[i] = horseList.get(i);
   }
}
```

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`

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--;
    }
}
```