2025

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free-Response Question 3

- ☑ **Scoring Guidelines**
- ☑ **Student Samples**
- ☑ **Scoring Commentary**

Applying the Scoring Criteria

Apply the question scoring criteria 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., printing to output, incorrect precondition check)
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., valid 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 counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

Question 3: Array / ArrayList — 9 points

Canonical solution

a. **4 points**

```
public Round(String[] names)
{
   competitorList = new ArrayList<Competitor>();
   for(int i = 0; i < names.length; i++)
   {
      Competitor c = new Competitor(names[i], i + 1);
      competitorList.add(c);
   }
}
```

b. **5 points**

```
public ArrayList<Match> buildMatches()
{
   ArrayList<Match> matches = new ArrayList<Match>();

   if (competitorList.size() % 2 == 0)
   {
      for(int i = 0; i < competitorList.size()/2; i++)
      {
         matches.add(new Match(competitorList.get(i),
               competitorList.get(competitorList.size() – i - 1)));
      }
   }
   else
   {
      for(int i = 1; i < competitorList.size() / 2 + 1; i++)
      {
         matches.add(new Match(competitorList.get(i),
             competitorList.get(competitorList.size() - i)));
      }
   }
   return matches;
}
```

a. `Round`

	Scoring Criteria	Decision Rules	
1	Accesses* all elements of `names` (*no bounds errors*)	Responses **will not** earn the point if they • access `names` incorrectly	**1 point**
2	Initializes and maintains rank associated with each accessed competitor, beginning at 1	Responses **can** still earn the point even if they • maintain a rank variable with initial value 0, as long as it is offset by 1 when accessed • fail to construct a `Competitor` object Responses **will not** earn the point if they • compute rank incorrectly for any competitor	**1 point**
3	Constructs `Competitor` with provided name and computed rank	Responses **can** still earn the point even if they • access names incorrectly • compute rank incorrectly Responses **will not** earn the point if they • omit `new` in the construction of a `Competitor` or fail to use the correct number and type of parameters	**1 point**
4	Adds all constructed competitors to `competitorList` in the correct order (*algorithm*)	Responses **can** still earn the point even if they • fail to initialize `competitorList` *(`ArrayList` creation not assessed in this part)* • omit `new` in the construction of a `Competitor` • fail to access all elements of `names` • add elements with an incorrect or missing rank Responses **will not** earn the point if they • add items without constructing a `Competitor` • access `competitorList` incorrectly • fail to update the instance variable `competitorList` (e.g. by redeclaring as a local variable) • print or return a value instead of or in addition to updating `competitorList`	**1 point**

*An enhanced `for` loop inherently accesses all elements of an `ArrayList`

b. `buildMatches`

	Scoring Criteria	Decision Rules	
5	Declares and initializes local `ArrayList` of `Match` objects	Responses **will not** earn the point if they • fail to declare the `ArrayList`, even if they have other local variables declared	**1 point**
6	Initializes and maintains an index used to move from both ends of `competitorList` *(algorithm)*	Responses **can** still earn the point even if they • start at 0 instead of 1 or vice versa • make a bounds error with the "high" index, as long as it is counting down from the end of the list • maintain two separate index variables instead of a single offset or other equivalent strategy, as long as it is used to count up from the start and down from the end • fail to use the indices to construct a `Match`	**1 point**
7	Computes starting low index based on even/odd comparison	Responses **can** still earn the point even if they • compute the index incorrectly, as long as the even/odd cases start at different indices • call the `size` method incorrectly • modifies `competitorList` instead of skipping an index Responses **will not** earn the point if they • determine even and odd number of competitors incorrectly	**1 point**
8	Gets two elements of `competitorList` based on maintained index, constructs a `Match` object from them, and adds it to the local list	Responses **can** still earn the point even if they • omit `new` in the creation of the `Match` object *(use of `new` not assessed for this type)* • use incorrect indices Responses **will not** earn the point if they • access `competitorList` incorrectly • fail to create an object of type `Match` • use incorrect number or type of parameters on any of the method calls	**1 point**

9	Populates the list with the correct number of pairs of competitors, omitting the first competitor in the odd case, without bounds errors *(algorithm)*	Responses **can** still earn the point even if they • fail to return the `ArrayList` (*return not assessed in this question*) • determine even and odd number of competitors incorrectly, as long as the cases are unambiguous • fail to create or use a `Match` object Responses **will not** earn the point if they • fail to guard against even and odd numbers of competitors • include the first competitor for odd numbers of competitors or omit the first for even numbers • include too many matches (e.g. due to continuing past the middle of the list) • add indices instead of competitors • make syntactically incorrect call(s) to `size` or other `ArrayList` methods • permanently modify `competitorList`	**1 point**
	Question-specific penalties		
	None		

Alternate canonical solution:

```
public Round(String[] names)
{
    competitorList = new ArrayList<Competitor>();

    int rank = 1;

    for (String name : names)
    {
       Competitor c = new Competitor(name, rank);
       competitorList.add(c);
       rank++;
    }
}


public ArrayList<Match> buildMatches()
{
   ArrayList<Match> matches = new ArrayList<Match>();

   int low = 0;
   if (competitorList.size() % 2 == 1)
   {
      low = 1;
   }
   int high = competitorList.size() - 1;

   while (low < high)
   {
      Match m = new Match(competitorList.get(low),
                          competitorList.get(high));
      matches.add(m);
      low++;
      high--;
   }
   return matches;

}
```

Part (a)

```
public Round(String[] names){
   competitorList = new ArrayList<Competitor>();
   for(int i = 0; i<names.length; i++){
      competitorList.add(new Competitor(names[i], i+1));
   }
}
```

Part (b)

```
public ArrayList<Match> buildMatches(){
   ArrayList<Match> matches = new ArrayList<Match>();
   int k = 0;
   if(competitorList.size() % 2 != 0){
      k = 1;
   }
   for(int i = k; i<(competitorList.size()-k)/2; i++){
      matches.add(competitorList.get(i),
                  competitorList.get(competitorList.size()-1-i+k));
   }
   return matches;
}
```

Part (a)

```
competitorList = new ArrayList<>();

for(int i =0; i<names.length; i++){
   competitorList.add(new Competitor(names[i], i));
}
```

Part (b)

```
ArrayList<Match> games = new ArrayList<Match>();
int i = 0;

if(competitorsList.size() % 2 == 0){
   while(i!=competitorsList.size()/2){
      games.add(new Match(competitorsList.get(i),
                  competitorsList.get(competitorsList.size()-1-i)));
      i++;
   }
}else{
   while(i!=(competitorsList.size()-1)/2){
      games.add(new Match(competitorsList.get(i),
                   competitorsList.get(competitorsList.size()-1-i)));
      i++;
   }
}
```

Part (a)

```
public Round(String[] names) {
   ArrayList<String> competitorList = new ArrayList<>();
   for (int i = 0; i < names.length(); i++) {
      competitorList.add(names[i]);
   }
}
```

Part (b)

```
public ArrayList<Match> buildMatches() {
   if (competitorList.size() % 2 == 0) {
      for (int i = 0; i < competitorList.size() / 2; i++) {
         buildMatches.add(competitorList.get(i) +
                          competitorList.get(competitorList.size() - i));
      }
   else {
      for (int i = 1; i < competitorList.size() / 2; i++) {
         buildMatches.add(competitorList.get(i) +
                          competitorList.get(competitorList.size() - i));
      }
   }
}
```

Question 3

Note: Student samples are quoted verbatim and may contain spelling and grammatical errors.

Overview

NEW for 2025: The question overviews can be found in the *Chief Reader Report on Student Responses* on AP Central.

Sample: 3A
Score: 7

In part (a) point 1 was earned by using a `for` loop that starts at the first element of the `names` array and accesses all the elements with `names[i]` without going out of bounds using `names.length` for the end condition. Point 2 was earned because the response initializes and maintains a correct rank for each accessed competitor beginning at `1`. The loop variable `i` begins at `0` and the response uses `i+1` to ensure the first rank is not zero. Point 3 was earned because the response correctly constructs a new `Competitor` object with the provided name as `names[i]` and computed rank as `i+1` and the correct use of the `new` keyword. Point 4 was earned because all constructed competitors are added to `competitorList` in the correct order by adding the new `Competitor` object to the list using `competitorList.add()`.

In part (b) point 5 was earned because the response correctly declares and initializes a local `ArrayList<Match>`, called `matches`, without syntax errors. However, for Point 5 the `ArrayList<Match>` object must be constructed. Point 6 was earned because the response appropriately initializes and maintains a "low" and "high" index. The loop variable `i` is used as the index of the "low" end of the list and the "high" end of the list is calculated using `(competitorList.size() - 1 - i + k)`. During the `for` loop traversal, the "low" index is incremented to move up from the beginning of `competitorList` and the "high" index in decremented to move down from the end of `competitorList`. Point 7 was earned because the response correctly uses the remainder operator (`%`) to determine whether the size of `competitorList` is odd (`if(competitorList.size() % 2 != 0)`), then adjusts the starting "low" index appropriately. The response uses variable `k` beginning at `0` and changes it to a value of `1` if the condition indicates this list has an odd number of elements. The loop control variable `i` is initialized to `k` to determine the starting "low" index. Point 8 was not earned because the response does not create a `match` object. The response correctly gets two elements of `competitorList` using the maintained index values but incorrectly attempts to add them to `games` using the `add` method. Point 9 was not earned because the response does not populate the list with the correct number of pairs in all cases. When `competitorList` contains an odd number of elements and the value `k` is set to `1`, the `for` loop should increment from index `1` up to (and including) the middle index position. The terminating condition uses `<` instead of `<=`. As a result, the loop terminates early, omitting one pair of competitors from the list. It should be noted that although Point 8 was not earned due to a `Match` object not being created, Point 9 can still be earned if the response fails to create or use a `Match` object.

Question 3 (continued)

Sample: 3B
Score: 6

In part (a) point 1 was earned because all elements of `names` are accessed with no bounds errors by using `names.length` in the `for` loop and accessing each element with `names[i]`. Point 2 was not earned because the response incorrectly uses the loop control variable `i` to represent rank without adjusting it to start with a rank of `1` instead of `0`. Point 3 was earned because the response correctly constructs a `Competitor` object with the provided name and rank with the statement `new Competitor(names[i], i)`. The incorrect initialization of the provided rank was assessed in Point 2. Point 4 was earned because the response correctly adds all constructed competitors to `competitorList` in the correct order. Note that this point is still earned even though the rank was incorrect.

In part (b) point 5 was earned because the response correctly declares and initializes a local `ArrayList` of `Match` objects called `games`, without syntax errors. Point 6 was earned because the response initializes and maintains the variable `i` to move from both ends of `competitorList`. In both the even and odd cases, the variable `i` is used to move from the front of the list toward the back, and `competitorsList.size()-1-i` is used to move from the back of the list toward the front. The error of starting at `0` for both the even and odd cases will be assessed in Point 7. It should be noted that the use of `competitorsList` instead of `competitorList` is considered a spelling discrepancy and is one of the minor errors for which no penalty is assessed, as noted on the *Applying the Scoring Criteria* page found in the Scoring Guidelines. Point 7 was not earned because the response incorrectly begins the starting low index at `0` for both the even and odd cases. Point 8 was earned because the response correctly gets two elements from `competitorList` based on maintained index values, uses them to construct a `Match` object, and then adds the `Match` object to `games`. Point 9 was not earned because, even though the `ArrayList` named `games` is populated with the correct number of pairs of matches, the competitor in index position `0` is included in the odd case. Failure to return the `ArrayList` is not assessed in this question.

Sample: 3C
Score: 3

In part (a) point 1 was earned because all elements of `names` are accessed with no bounds errors by using `names.length` in the `for` loop and accessing each element with `names[i]`. Point 2 was not earned because the response fails to create or maintain a variable for the rank associated with each accessed competitor. Point 3 was not earned because the response does not construct a `Competitor` object. Point 4 was not earned for the following two reasons, either of which would cause the response to lose Point 4. The response attempts to add items to `competitorList` without constructing a `Competitor` object. It should be noted that, had the response tried to construct a `Competitor` object and omitted `new` this point could still be earned. Furthermore, the response incorrectly declares `competitorList` as a local variable. As a result, the instance variable `competitorList` is never updated.

Question 3 (continued)

In part (b) point 5 was not earned because the response does not declare and initialize a local `ArrayList` of `Match` objects. Point 6 was earned because the response correctly maintains values that are used to move from both ends of `competitorList`. The response uses the `for` loop control variable `i` to move towards the end of the list and `competitorList.size()-i` to move toward the front of the list. It should be noted that a bounds error with the index variables is allowed provided they are used to count up from the start and down from the end. Point 7 was earned because the response correctly uses the remainder (`%`) operator to determine whether the size of `competitorList` is even or odd, then adjusts the starting low index appropriately by initializing the loop control variable to `0` for the even case or `1` for the odd case. Point 8 was not earned because the response does not create a `Match` object. The response correctly gets two elements of `competitorList` using the maintained index values but incorrectly attempts to add them to an `ArrayList` called `buildMatches`, which is not defined, using an incorrect call to the `add` method. Note this response has not defined a list to store the matches. This is assessed in point 9. Additionally, the response uses an addition operator instead of a comma to delineate the two elements. Point 9 was not earned for multiple reasons, any of which would cause the response to lose Point 9. First, the `for` loop contains a bounds error. When `i` is `0` (in the even case), the call `competitorList.get(competitorList.size() - i)` will be out of bounds. Second, the response never populates a local ArrayList with matches (`buildMatches` is the name of the method, not a defined list). Finally, in the odd case, the response adds one too few matches because the `for` loop conditional should use `<=` instead of `<`.