

2024



---

# AP<sup>®</sup> 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 (`x • ÷ ≤ ≥ <> ≠`)
- `[]` 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.*

**Question 3: Array / ArrayList****9 points****Canonical solution**

- (a) `public boolean isWordChain()` **3 points**
- ```
{
    for (int i = 1; i < wordList.size(); i++)
    {
        String current = wordList.get(i);
        String previous = wordList.get(i - 1);

        if (current.indexOf(previous) == -1)
        {
            return false;
        }
    }
    return true;
}
```
- (b) `public ArrayList<String> createList(String target)` **6 points**
- ```
{
    ArrayList<String> result = new ArrayList<String>();

    for (String current : wordList)
    {
        if (current.indexOf(target) == 0)
        {
            String newStr = current.substring(target.length());
            result.add(newStr);
        }
    }

    return result;
}
```

**(a)** `isWordChain`

Scoring Criteria		Decision Rules	
<b>1</b>	Accesses all adjacent pairs of <code>wordList</code> elements ( <i>no bounds errors</i> )	<p>Responses <b>can</b> still earn the point even if they</p> <ul style="list-style-type: none"> <li>also access non-adjacent pairs of elements</li> <li>return early, as long as bounds and indices would otherwise support accessing all necessary pairs</li> </ul> <p>Responses <b>will not</b> earn the point if they</p> <ul style="list-style-type: none"> <li>fail to access elements of <code>wordList</code> correctly</li> </ul>	<b>1 point</b>
<b>2</b>	Determines whether an element of the list contains a previous element of the list	<p>Responses <b>can</b> still earn the point even if they</p> <ul style="list-style-type: none"> <li>make just one comparison</li> <li>fail to make the comparison in the context of a loop</li> <li>compare every element to the first element of the list</li> <li>access pairs of <code>wordList</code> elements incorrectly</li> </ul> <p>Responses <b>will not</b> earn the point if they</p> <ul style="list-style-type: none"> <li>make an incorrect call to <code>indexOf</code> or use the <code>indexOf</code> return value incorrectly</li> </ul>	<b>1 point</b>
<b>3</b>	Returns appropriate <code>boolean</code> in both cases ( <i>algorithm</i> )	<p>Responses <b>can</b> still earn the point even if they</p> <ul style="list-style-type: none"> <li>incorrectly identify whether an element of <code>wordList</code> contains the previous element</li> </ul> <p>Responses <b>will not</b> earn the point if they</p> <ul style="list-style-type: none"> <li>return an incorrect value due to an early return</li> <li>fail to return <code>true</code> or <code>false</code></li> </ul>	<b>1 point</b>
<b>Total for part (a)</b>			<b>3 points</b>

**(b)** `createList`

Scoring Criteria		Decision Rules	
<b>4</b>	Declares and constructs an <code>ArrayList&lt;String&gt;</code>	Responses <b>will not</b> earn the point if they <ul style="list-style-type: none"> <li>fail to declare an <code>ArrayList</code></li> </ul>	<b>1 point</b>
<b>5</b>	Accesses all elements of <code>wordList</code> ( <i>no bounds errors</i> )	Responses <b>can</b> still earn the point even if they <ul style="list-style-type: none"> <li>return early, as long as bounds and indices would otherwise support accessing all elements</li> </ul> Responses <b>will not</b> earn the point if they <ul style="list-style-type: none"> <li>fail to access elements of <code>wordList</code> correctly</li> </ul>	<b>1 point</b>
<b>6</b>	Identifies strings that begin with <code>target</code> ( <i>in the context of an <code>if</code></i> )	Responses <b>can</b> still earn the point even if they <ul style="list-style-type: none"> <li>access elements of <code>wordList</code> incorrectly</li> </ul> Responses <b>will not</b> earn the point if they <ul style="list-style-type: none"> <li>call <code>String</code> methods incorrectly</li> <li>identify all strings that contain <code>target</code></li> <li>use the <code>substring</code> method without a guard against an element too short to contain <code>target</code></li> </ul>	<b>1 point</b>
<b>7</b>	Constructs a <code>String</code> that is a copy of an element of the list with the correct number of initial characters removed	Responses <b>can</b> still earn the point even if they <ul style="list-style-type: none"> <li>make a copy of a <code>wordList</code> element that does not start with <code>target</code> or is not long enough to contain <code>target</code></li> </ul>	<b>1 point</b>
<b>8</b>	Adds to the constructed list at least one <code>String</code> based on an element of the original list	Responses <b>can</b> still earn the point even if they <ul style="list-style-type: none"> <li>add an incorrectly constructed <code>String</code></li> <li>have not constructed a list</li> </ul> Responses <b>will not</b> earn the point if they <ul style="list-style-type: none"> <li>call <code>add</code> incorrectly</li> </ul>	<b>1 point</b>

<b>9</b>	Returns list containing all and only identified and revised strings in the appropriate order ( <i>algorithm</i> )	Responses <b>can</b> still earn the point even if they <ul style="list-style-type: none"><li>• incorrectly identify strings beginning with <code>target</code></li><li>• call <code>add</code> incorrectly</li></ul> Responses <b>will not</b> earn the point if they <ul style="list-style-type: none"><li>• add the original, unrevised element to the list to be returned</li><li>• modify <code>wordList</code> or any of its elements</li><li>• return an incorrect value due to an early return</li></ul>	<b>1 point</b>
		<b>Total for part (b)</b>	<b>6 points</b>
		<b>Total for question 3</b>	<b>9 points</b>

Note that a correct part (b) solution could replace the `indexOf` call in the `if` statement with:

```
if (current.length() >= target.length() &&
    current.substring(0, target.length()).equals(target))
```

- Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1

Question 2

Question 3

Question 4



Begin your response to each question at the top of a new page.

a

```
public boolean isWordChain()
    String chain = wordList.get(0);
    for (int i = 1; i < wordList.size(); i++)
        if (wordList.get(i).indexOf(chain) != -1)
            chain = wordList.get(i);
        else
            return false;
    return true;
```

b

```
public ArrayList<String> createList(String target)
    ArrayList<String> removed = new ArrayList<String>();
    int targetLen = target.length();
    for (int i = 0; i < wordList.size(); i++)
        if (wordList.get(i).substring(0, targetLen)
            .equals(target))
            removed.add(wordList.get(i).substring(
                0, targetLen));
    return removed;
```

Page 5

Use a pencil only. Do NOT write your name. Do NOT write outside the box.

0024849



● **Important:** Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1

Question 2

Question 3

Question 4



Begin your response to each question at the top of a new page.

```

Public boolean isWordChain()
{
    public boolean ans;
    for (int x = 1; x <= wordList.size() - 1; x++)
    {
        for (int y = 0; y <= wordList[x].length; y++)
        {
            if (wordList[x-1].equals(wordList[x].substring(y, wordList[x-1].length))
            {
                ans = true;
            }
            else
            {
                ans = false;
            }
        }
    }
    return ans;
}
    
```

Use a pencil only. Do NOT write your name. Do NOT write outside the box.

0118448





Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1



Question 2



Question 3



Question 4



Begin your response to each question at the top of a new page.

```
public ArrayList<String> createList (String target)
{
    ArrayList<String> correct = new ArrayList<>();
    for (int x=0; x < wordList.size(); x++)
    {
        if (target.equals (wordList [x].substring (0, target.length)))
        {
            correct.add (wordList [x].substring (target.length));
        }
    }
    return correct;
}
```

Use a pencil only. Do NOT write your name. Do NOT write outside the box.

● Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1

Question 2

Question 3

Question 4



Begin your response to each question at the top of a new page.

```
a) public boolean isWordChain()
{
  for(int i=1; i< wordlist.size()-1; i++)
  {
    if(wordlist.get(i).indexOf(wordlist.get(i-1).substring(1))
    != -1)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
}
```

Use a pencil only. Do NOT write your name. Do NOT write outside the box.

0057694



Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1

Question 2

Question 3

Question 4



Begin your response to each question at the top of a new page.

b)

```
public ArrayList<String> createList(String target)
```

```
{
    ArrayList<String> list = new ArrayList<>();
```

```
    for(int i = 0; i < wordList.size(); i++)
```

```
        if(wordList.get(i).substring(0, target.length())
            .equals(target))
```

```
        {
            list.add(wordList.get(i).substring(target.length()
                + 1));
        }
```

```
        if(!wordList.get(i).equals(target))
```

```
            list.add("");
```

```
    }
    return list;
```

```
}
```

Page 6

Use a pencil only. Do NOT write your name. Do NOT write outside the box.

### Question 3

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

#### Overview

This question tested the student’s ability to:

- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).
- Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects (Skill 3.D).

Students were asked to write two methods of a `WordChecker` class. The class contains an `ArrayList` of `String` objects that is already populated. Students were expected to traverse and manipulate the existing `ArrayList` and construct a new one, and to access and manipulate `String` values.

In part (a) students were asked to write a method that determines whether the stored list represents a “word chain.” Doing so requires a traversal capable of accessing all adjacent pairs in the list and comparing those adjacent strings, generally using `indexOf` to determine whether an element of the list contained a previous element. Looking for pairs that do *not* have the property, the method must return `false` when it finds an invalid pair (and `true` at the end if none of the pairs are invalid).

In part (b) students were asked to write an unrelated method. This method needs to construct, build, and return a new `ArrayList` with contents derived from the stored list. To do so, the method must traverse the `wordList` and check if each string element starts with a target string (using either `indexOf` or a guarded call to `substring`), and if so, extract the remainder of the string and add the result to the constructed `ArrayList`.

#### Sample: 3A

##### Score: 8

In part (a) point 1 was earned by using a `for` loop that starts at the second element of the list and compares it with the previous element (adjacent), which was stored in the `chain` variable prior to the `for` loop. The response correctly accesses the elements of `wordList` using the `get` method. The `chain` variable is updated to store the currently accessed element in `wordList` using the `get` method. On subsequent iterations of the `for` loop, the next element is accessed and compared to the previous element stored in `chain`. Additionally, the `for` loop is written correctly and uses the local variable `i` to access the current element twice within the loop. Point 2 was earned because it compares an element of `wordList` with a previous element of `wordList` using the `String` `indexOf` method and checks whether the value returned is not equal to `-1`. Point 3 was earned by returning the correct `boolean` value in both cases. If a pair is found that does not meet the criterion in the conditional statement, the response immediately returns `false`. Otherwise, the response compares subsequent pairs in `wordList`. When all pairs meet the criterion, the response returns `true`.

**Question 3 (continued)**

In part (b) point 4 was earned because the response correctly declares and constructs an `ArrayList` variable, `removed`. Point 5 was earned by using a `for` loop to traverse all elements of `wordList`. The response uses the `get` method and the element's index to access the element correctly. Point 6 was not earned because the response does not guard against the length of the target being greater than the length of the element from the `ArrayList`. Thus, the `substring` method call throws an `IndexOutOfBoundsException` exception when the length of `target` is greater than the length of a `wordList` element. The response does correctly check that the `String` begins with `target`. Point 7 was earned by constructing a `String` that has `target` removed from the beginning of an element of `wordList`, using the one-parameter `substring` method. The potential out-of-bounds error is assessed in point 6 and is not assessed here. Point 8 was earned by correctly calling the `add` method with a `String` argument based on an element of `wordList`. This point focuses exclusively on adding a `String` based on an element of the original list to the constructed list. Point 9 was earned by returning the list that contains all and only identified and revised strings in the appropriate order.

**Sample: 3B****Score: 4**

In part (a) point 1 was not earned because the response uses array syntax instead of `ArrayList` syntax to access adjacent pairs of `wordList` elements. Because this is assessed in the rubric, penalty point (v) from the “1-Point Penalty” list in the Scoring Guidelines is not applied. Point 2 was not earned because the response does not determine whether an element of the list contains a previous element of the list. The response uses an inner loop to attempt to find an occurrence of the previous element within the current element, but the `substring` method call `wordList[x].substring(y, wordList[x - 1].length)` could result in an out-of-bounds error. Note that the response can still earn this point even if it accesses pairs of `wordList` elements incorrectly. Point 3 was not earned because the response does not return the appropriate `boolean` in both cases. The `boolean ans` is assigned its value based on only the last iteration of the loop before it is returned.

In part (b) point 4 was earned because the response declares and constructs an `ArrayList<String>`. The missing parentheses in the `ArrayList` constructor invocation is one of the minor errors for which no penalty is assessed on this exam. (See the “No Penalty” section of the Scoring Guidelines for a complete list.) Point 5 was not earned because the response uses array syntax instead of `ArrayList` syntax to access elements of `wordlist`. Point 6 was not earned because the response calls `wordList[x].substring(0, target.length)` without a guard against an element shorter than `target`. The missing parentheses on the no-parameter `length` method is one of the minor errors for which no penalty is assessed. Point 7 was earned because the response constructs a `String` that is a copy of an element of the list with the correct number of initial characters removed. The use of array syntax to access elements of `wordList` does not affect whether this point is earned. Point 8 was earned because the response correctly adds to the constructed list at least one `String` based on an element of the original list. Point 9 was earned because the response returns a list containing all and only identified and revised strings in the appropriate order.

**Question 3 (continued)****Sample: 3C****Score: 3**

In part (a) point 1 was not earned because the response does not access all adjacent pairs of `wordList` elements. The response accesses elements at indices `i` and `i - 1` in the loop, and `i` never represents the last index in `wordList` because the conditional part of the `for` loop header is `i < wordList.size() - 1`. Point 2 was earned because the response properly uses `indexOf` to determine whether an element of the list contains a previous element of the list. Point 3 was not earned because there is an early return of `true` within the loop.

In part (b) point 4 was not earned because the response does not correctly construct an `ArrayList<String>`. The response incorrectly uses `new <String>` without the `ArrayList` constructor. Point 5 was earned because the response uses a correct `for` loop and the `get` method to access all elements of `wordList` without bounds errors. Point 6 was not earned because the response calls `wordList.get(i).substring(0, target.length())` without a guard against an element shorter than `target`. Point 7 was not earned because the response constructs a `String` with the incorrect number of initial characters removed. The `substring` parameter is `target.length()+1` instead of `target.length()`. Point 8 was earned because the response adds to the constructed list at least one `String` based on an element of the original list. Point 9 was not earned because the response returns a list containing more strings than it should. Whenever `target` matches a `wordList` element, two empty strings, `""`, are added to the returned list because both `if` statements execute.