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.*

AP® COMPUTER SCIENCE A
2016 SCORING GUIDELINES

Question 1: Random String Chooser

Part (a)	`RandomStringChooser`	**7 points**

Intent: *Define implementation of class to choose a random string*

+1 Uses correct class, constructor, and method headers

+1 Declares appropriate `private` instance variable(s)

+1 Initializes all instance variable(s) (*point lost if parameter not used in any initialization*)

+4 Implements `getNext`

- **+1** Generates a random number in the proper range (*point lost for improper or missing cast*)
- **+1** Chooses a string from instance variable using generated random number
- **+1** Updates state appropriately (*point lost if constructor parameter is altered*)
- **+1** Returns chosen string or `"NONE"` as appropriate

Part (b)	`RandomLetterChooser`	**2 points**

Intent: *Define implementation of a constructor of a class that extends* `RandomStringChooser`

+1 `getSingleLetters(str)`

+1 `super(getSingleLetters(str));` (*point lost if not first statement in constructor*)

AP® COMPUTER SCIENCE A
2016 CANONICAL SOLUTIONS

Question 1: Random String Chooser

Part (a):

```
public class RandomStringChooser
{
    private List<String> words;

    public RandomStringChooser(String[] wordArray)
    {
       words = new ArrayList<String>();

       for (String singleWord : wordArray)
       {
            words.add(singleWord);
       }
    }

    public String getNext()
    {
       if (words.size() > 0)
       {
            return words.remove((int)(Math.random() * words.size()));
       }
       return "NONE";
    }
}
```

Part (b):

```
     public RandomLetterChooser(String str)
     {
          super(getSingleLetters(str));
     }
```

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.

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

```
public class RandomStringChooser
{
  private ArrayList<String> words;
  public RandomStringChooser(String[] array)
  {
      words = new ArrayList<String>;
      for(int i=0; i<array.length; i++)
      {
          words.add(array[i]);
      }
  }
  public String getNext()
  {
     if (words.size() == 0)
     return "NONE";
int rand = (int)(Math.Random() * words.length());
     String temp = words.get(rand);
      words.remove(rand);
       return temp;
  }
}
```

Part (b) begins on page 6.

1Ab

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 *  Precondition: str contains only letters.
 */
public RandomLetterChooser(String str)
{
    super( getSingleLetters(str);
}
```

1B9

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

```
public class RandomStringChooser
{
  private String[] str;

  public RandomStringChooser(String[] arr)
  {
    str = arr;
  }

  public String getNext()
  {
    String result = "";
    int x = Math.random() * str.length;
    return str[x]
```

Part (b) begins on page 6.

1Bb

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 *  Precondition: str contains only letters.
 */
public RandomLetterChooser(String str)
{ String[] x = getSingleLetters(str);
}
```

1Ca

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

```
PUBLIC CLASS RANDOMSTRINGCHOOSER(){
  PRIVATE STRING[] WORDARR;
  PUBLIC RANDOMSTRINGCHOOSER(STRING[] TEST){
    WORDARR = TEST;
  }
  PUBLIC STRING GETNEXT(STRING[] TEST){
    INT K = TEST.LENGTH;
    INT COUNT = 0;
    WHILE (COUNT ≤ K){
      RETURN TEST[MATH.RANDOM()* K];
      COUNT ++;
    }
  RETURN "NONE";
  }
}
```

Part (b) begins on page 6.

1Cb

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 *  Precondition: str contains only letters.
 */
public RandomLetterChooser(String str) {
    STRING[] ANS = STR.getSingleLetters(STR);
}
```

AP® COMPUTER SCIENCE A
2016 SCORING COMMENTARY

Question 1

Overview

This question focused on class design, inheritance, the use of the array data structure, and the accessing of elements from that array.

In part (a) students were asked to write the complete `RandomStringChooser` class. Students had to demonstrate an understanding of class, constructor, and method header syntax. Students were expected to choose appropriate instance variable(s) to maintain object state, declaring those instance variables `private`, and initializing the instance variables in a constructor. Students needed to include in their class a `getNext` method, which returns a randomly chosen `String` from the array argument, taking care to not return a particular `String` more than once. Two approaches work equally well here: making a second array to help keep track of used elements or copying all of the array elements into an `ArrayList` and removing elements from that list after they are used. Students were explicitly told not to alter the parameter passed to the constructor; therefore, the choice of data structure was significant. Students were also required to demonstrate an understanding of how to generate a random number in the proper range for the purposes of selecting a `String` from the array.

In part (b) students were asked to the write the constructor for the `RandomLetterChooser` class, which is a subclass of the `RandomStringChooser` class they wrote in part (a). Students needed to know how to invoke the constructor from the superclass using `super`. Students were given access to the static `getSingleLetters` method and were explicitly told to use it for full credit rather than reimplement the functionality the method provides. Students were expected to call the `getSingleLetters` method appropriately to create an array of `Strings` from the given `String` parameter.

Sample: 1A
Score: 9

In part (a) the student correctly creates the class, constructor, and the `getNext` method headers, which earned the first point. The student declares an `ArrayList` reference as a `private` instance variable, which earned the second point. Point 3 was earned by instantiating the `ArrayList` (no penalty for the missing `()` on the zero-parameter constructor call) and populating the `ArrayList` with values from the parameter. The student correctly generates a random number in the proper range, which earned point 4. The student chooses a `String` from the instance variable based on the number the student randomly generated, which earned point 5. The student updates the instance variable without altering the given parameter, which earned point 6. Point 7 was earned by returning the chosen `String` or `"None"` as appropriate. Part (a) earned 7 points.

In part (b) the student demonstrates knowledge that `super` has to come first in a subclass constructor and passes the correct call to `getSingleLetters` to `super`, which earned the final points. Part (b) earned 2 points.

Sample: 1B
Score: 5

In part (a) the student correctly creates the class, constructor, and the `getNext` method headers, which earned the first point. The student declares a reference to an array of `Strings` as a `private` instance variable, which earned the second point. Point 3 was earned by initializing the array correctly in the constructor using the given parameter. The student does not correctly generate a random number in the

Question 1 (continued)

proper range due to the lack of a cast to `int` and did not earn point 4. The student chooses a `String` from the instance variable using proper array syntax based on the randomly generated number and earned point 5. The student does not make an attempt to update state, so point 6 was not earned. Point 7 was not earned because `"None"` was not returned as appropriate. Part (a) earned 4 points.

In part (b) point 8 was earned by a correct call to `getSingleLetters` with the correct parameter. Because there is no call to `super`, the student did not earn the final point. Part (b) earned 1 point.

Sample: 1C
Score: 2

In part (a) the student does not correctly create the `getNext` method header because the student includes an argument, which did not earn the first point. The student declares a reference to an array of `Strings` as a `private` instance variable, which earned the second point. Point 3 was earned by initializing the array correctly in the constructor using the given parameter. The student does not correctly generate a random number in the proper range due to the lack of a cast to `int` and did not earn point 4. The student did not earn point 5 because the `String` the student chose based on the randomly generated number was from the parameter and not the instance variable. The student does not make an attempt to update state, so point 6 was not earned. Point 7 was not earned because there is a chance a particular `String` could be returned multiple times as there is no attempt to reduce the choices available. Part (a) earned 2 points.

In part (b) the student does not call `getSingleLetters` with the correct parameter, so point 8 was not earned. Because there was no call to `super`, the student did not earn the final point. Part (b) did not earn any points.