AP® COMPUTER SCIENCE A
2017 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., 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
o) Extraneous code with no side-effect (e.g., valid precondition check, no-op)
o) Spelling/case discrepancies where there is no ambiguity*
o) Local variable not declared provided other variables are declared in some part
o) private or public qualifier on a local variable
o) Missing public qualifier on class or constructor header
o) Keyword used as an identifier
o) Common mathematical symbols used for operators (× • ÷ < > ≠)
o) [ ] vs. ( ) vs. <>
o) = instead of == and vice versa
o) length/size confusion for array, String, List, or ArrayList; with or without()
o) Extraneous [ ] when referencing entire array
o) [i, j] instead of [i][j]
o) Extraneous size in array declaration, e.g., int[size] nums = new int[size];
o) Missing ; where structure clearly conveys intent
o) Missing { } where indentation clearly conveys intent
o) Missing ( ) on parameter-less method or constructor invocations
o) 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 “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 1: Digits

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>Digits constructor</th>
<th>5 points</th>
</tr>
</thead>
</table>

**Intent:** Initialize instance variable using passed parameter

+1 Constructs digitList

+1 Identifies a digit in num

+1 Adds at least one identified digit to a list

+1 Adds all identified digits to a list (*must be in context of a loop*)

+1 **On exit:** digitList contains all and only digits of num in the correct order

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>isStrictlyIncreasing</th>
<th>4 points</th>
</tr>
</thead>
</table>

**Intent:** Determine whether or not elements in digitList are in increasing order

+1 Compares at least one identified consecutive pair of digitList elements

+1 Determines if a consecutive pair of digitList is out of order (*must be in context of a digitList traversal*)

+1 Compares all necessary consecutive pairs of elements (*no bounds errors*)

+1 Returns true iff all consecutive pairs of elements are in order; returns false otherwise

**Question-Specific Penalties**

-2 (q) Uses confused identifier instead of digitList
## AP® COMPUTER SCIENCE A
### 2017 SCORING GUIDELINES

### Question 1: Scoring Notes

#### Part (a) Digits constructor

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they ...</th>
<th>Responses will not earn the point if they ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Constructs digitList</td>
<td>• identify one digit of num or a length one substring/character of the String representation of num</td>
<td>• initialize a local variable instead of digitList</td>
</tr>
<tr>
<td>+1</td>
<td>Identifies a digit in num</td>
<td>• call add for some ArrayList using previously identified digit, even if that digit was identified incorrectly</td>
<td>• treat num itself as a String</td>
</tr>
<tr>
<td>+1</td>
<td>Adds at least one identified digit to a list</td>
<td>• call add for some ArrayList using previously identified digits, even if those digits were identified incorrectly</td>
<td>• identify only 1 digit</td>
</tr>
<tr>
<td>+1</td>
<td>Adds all identified digits to a list (must be in the context of a loop)</td>
<td>• add to digitList even if it is not instantiated properly</td>
<td>• obtain a list with the digits in reverse order</td>
</tr>
<tr>
<td></td>
<td>On exit: digitList contains all and only digits of num in the correct order</td>
<td>• add to digitList even if it is not instantiated properly</td>
<td>• omit one or more digits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• add extra digits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• mishandle edge case, e.g., 0 or 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• make a bounds error processing the String representation of num</td>
</tr>
</tbody>
</table>

#### Part (b) isStrictlyIncreasing

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point if they ...</th>
<th>Responses will not earn the point if they ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Compares at least one identified consecutive pair of digitList elements</td>
<td>• access digitList as an array or string</td>
<td>• access digitList as an array or string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fail to call .get()</td>
<td>• fail to call .get()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• compare using !&gt;</td>
<td>• compare using !&gt;</td>
</tr>
<tr>
<td></td>
<td>Determines if a consecutive pair of digitList is out of order (must be in context of a digitList traversal)</td>
<td>• determine the correct relationship between the two compared consecutive elements, even if the syntax of the comparison is incorrect</td>
<td>• fail to consider the case where the two elements are equal for the false case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• return early</td>
<td>• return early</td>
</tr>
<tr>
<td></td>
<td>Compares all necessary consecutive pairs of elements (no bounds errors)</td>
<td>• return prematurely via if (...) return false; else return true;</td>
<td>• return prematurely via if (...) return false; else return true;</td>
</tr>
<tr>
<td></td>
<td>Returns true if all consecutive pairs of elements are in order; returns false otherwise</td>
<td>• compare consecutive pairs for inequality, but fail to consider the case when two elements are equal</td>
<td>• compare consecutive pairs for inequality, but fail to consider the case when two elements are equal</td>
</tr>
</tbody>
</table>
Question 1: Digits

Part (a)

```java
public Digits(int num) {
    digitList = new ArrayList<Integer>();
    if (num == 0) {
        digitList.add(new Integer(0));
    }
    while (num > 0) {
        digitList.add(0, new Integer(num % 10));
        num /= 10;
    }
}
```

Part (b)

```java
public boolean isStrictlyIncreasing() {
    for (int i = 0; i < digitList.size()-1; i++) {
        if (digitList.get(i).intValue() >= digitList.get(i+1).intValue()) {
            return false;
        }
    }
    return true;
}
```

Note: The solutions shown above were written in compliance with the AP Java subset methods listed for Integer objects. Students were allowed to use the automatic "boxing" and "unboxing" of Integer objects in their solutions, which eliminates the need to use "new Integer (...)" in part (a) and "intValue ()" in part (b).
Complete the Digits constructor below.

```java
/** Constructs a Digits object that represents num.
 *  * Precondition: num >= 0
 */
public Digits(int num)
{
    List<Integer> list2 = new ArrayList<Integer>();
    digitList = new ArrayList<Integer>();
    if (num == 0) digitList.add(0);
    else
    {
        while (num != 0)
        {
            list2.add(num % 10);
            num /= 10;
        }
        for (int i = list2.size() - 1; i >= 0; i--)
        {
            digitList.add(list2.get(i));
        }
    }
}
```

Part (b) begins on page 6.
Complete method isStrictlyIncreasing below.

```java
/** Returns true if the digits in this Digits object are in strictly increasing order;
 *    false otherwise.
 */
public boolean isStrictlyIncreasing()
{
    boolean inc = true;
    for (int i = 1; i < digitList.size(); i++)
    {
        if (digitList.get(i - 1) >= digitList.get(i))
        {
            inc = false;
        }
    }
    return inc;
}
```
Complete the `Digits` constructor below.

```java
/** Constructs a Digits object that represents num.
 * Precondition: num >= 0
 */
public Digits(int num)
{
    int holder = num;
    String str = "";
    str += holder;
    for (int i = 0; i < str.length(); i++)
    {
        digitList.add(i, str.charAt(i));
    }
}
```

Part (b) begins on page 6.
Complete method `isStrictlyIncreasing` below.

```java
/** Returns true if the digits in this Digits object are in strictly increasing order; 
* false otherwise.
*/
public boolean isStrictlyIncreasing()
{
    int count = 0;
    for (int i = 0; i < digitList.size(); i++)
    {
        if (digitList.size() == 1)
            return true;
        else if (digitList[i] > digitList[i+1])
            count = 0;
        else if (digitList[i] < digitList[i+1])
            count++;
    }
    if (count >= digitList.size() - 1)
        return true;
    else return false;
}
```
Complete the `Digits` constructor below.

```java
/** Constructs a Digits object that represents num.
 *  * Precondition: num >= 0
 */
public Digits(int num) {
    String str = num.toString();
    for (int i = 0; i < str.length() - 1; i++) {
        String tempStr = str.substring(i, i + 1);
        int tempNum = Integer.parseInt(tempStr);
        Integer integer = new Integer(tempNum);
        digitList.add(integer);
    }
}
```

Part (b) begins on page 6.
Complete method isStrictlyIncreasing below.

```java
/** Returns true if the digits in this Digits object are in strictly increasing order;
 * false otherwise.
 */
public boolean isStrictlyIncreasing()
{
    boolean case = true;
    for (int i = digitList.size() - 1; i >= 0; i--)
    {
        if (digitList.get(i).intValue() <
            digitList.get(i - 1).intValue())
        {
            case = false;
        }
    }
    return case;
}
```
Question 1

Overview

This question involves identifying and processing the digits of a nonnegative integer. The `Digits` class is used to store these digits in an `ArrayList` private instance variable. Students were asked to write the constructor of the `Digits` class and a method to determine if the stored digits of the number are in strictly increasing order.

In part (a) students were asked to write a constructor that initializes the instance variable and fills it with all of the digits from the parameter `num`. Students must understand that the `ArrayList` instance variable must be instantiated before elements can be added. Students also needed to identify and isolate each digit of an integer value and add objects to an `ArrayList` in the correct order without going out of bounds.

In part (b) students were asked to complete a `boolean` method that returns `true` if the stored digits in `digitList` are in strictly increasing order; otherwise, it returns `false`. Students needed to demonstrate understanding of writing and evaluating `boolean` expressions and returning correct `boolean` values. Additionally, they needed to compare consecutive values in a list without going out of bounds.

Sample: 1A
Score: 9

In part (a) the response correctly instantiates the instance variable `digitList` and also declares and instantiates a temporary list. The response correctly identifies all digits of `num` and adds them to the temporary list in reverse order, meaning the last digit of `num` is placed in the first position of the temporary list, the next to last digit of `num` is placed in the second position of the temporary list, and so on. Then the response iterates through the temporary list from the end to the beginning and adds each digit to the end of `digitList`. On exit `digitList` contains all and only digits of `num` in the correct order. Part (a) earned 5 points.

In part (b) the logic correctly returns `true` when all consecutive pairs in `digitList` are strictly increasing; otherwise it returns `false`. Part (b) earned 4 points.

Sample: 1B
Score: 4

In part (a) point 1 was not earned because the response does not instantiate the instance variable `digitList`. The response correctly creates a `String` representation of `num` and then identifies one character of the string, which earned point 2. Point 3 was not earned because the response does not properly convert the identified character to a `Integer` before adding it to the list. The response earned point 4 for adding all digits to `digitList` and earned point 5 for maintaining the order of the digits as they appear in the parameter `num`. Part (a) earned 3 points.

In part (b) the response accesses `digitList` as an array and, therefore, did not earn point 1. Point 2 was not earned because the response fails to consider the case where two elements are equal. Point 3 was not earned because of a boundary error when comparing all consecutive pairs. The response earned point 4 by counting the number of elements that were in increasing order and then comparing the count to the size of the list - 1. This technique can be used to show that all pairs are strictly increasing. Part (b) earned 1 point.
Sample: 1C  
Score: 3

In part (a) point 1 was not earned because the response does not instantiate the instance variable `digitList`. Point 2 was not earned because the response incorrectly creates a `String` representation of `num`. Point 3 was not earned because the response does not properly convert the identified substring to an `Integer` before adding it to the list. The response earned point 4 by adding all identified digits to the list and did not earn point 5 because of a bounds error processing the `String` representation of `num`. Part (a) earned 1 point.

In part (b) the response compares two consecutive elements of `digitList` and earned point 1. Point 2 was not earned because the response fails to consider the case where two elements are equal. Point 3 was not earned due to a bounds error. Point 4 was earned because the logic correctly returns `true` when all consecutive pairs in `digitList` are strictly increasing; otherwise it returns `false`. Part (b) earned 2 points.