Question 1: Methods and Control Structures

This question involves the use of check digits, which can be used to help detect if an error has occurred when a number is entered or transmitted electronically. An algorithm for computing a check digit, based on the digits of a number, is provided in part (a).

The CheckDigit class is shown below. You will write two methods of the CheckDigit class.

```java
public class CheckDigit {
    /** Returns the check digit for num, as described in part (a).
     * Precondition: The number of digits in num is between one and six, inclusive.
     * num >= 0
     */
    public static int getCheck(int num) {
        // to be implemented in part (a)
    }

    /** Returns true if numWithCheckDigit is valid, or false otherwise, as described in part (b).
     * Precondition: The number of digits in numWithCheckDigit is between two and seven, inclusive.
     * numWithCheckDigit >= 0
     */
    public static boolean isValid(int numWithCheckDigit) {
        // to be implemented in part (b)
    }

    /** Returns the number of digits in num. */
    public static int getNumberOfDigits(int num) {
        // implementation not shown
    }

    /** Returns the nth digit of num.
     * Precondition: n >= 1 and n <= the number of digits in num
     */
    public static int getDigit(int num, int n) {
        // implementation not shown
    }

    // There may be instance variables, constructors, and methods not shown.
}
```
(a) Complete the `getCheck` method, which computes the check digit for a number according to the following rules.

- Multiply the first digit by 7, the second digit (if one exists) by 6, the third digit (if one exists) by 5, and so on. The length of the method’s `int` parameter is at most six; therefore, the last digit of a six-digit number will be multiplied by 2.
- Add the products calculated in the previous step.
- Extract the check digit, which is the rightmost digit of the sum calculated in the previous step.

The following are examples of the check-digit calculation.

**Example 1, where num has the value 283415**
- The sum to calculate is \((2 \times 7) + (8 \times 6) + (3 \times 5) + (4 \times 4) + (1 \times 3) + (5 \times 2) = 14 + 48 + 15 + 16 + 3 + 10 = 106\).
- The check digit is the rightmost digit of 106, or 6, and `getCheck` returns the integer value 6.

**Example 2, where num has the value 2183**
- The sum to calculate is \((2 \times 7) + (1 \times 6) + (8 \times 5) + (3 \times 4) = 14 + 6 + 40 + 12 = 72\).
- The check digit is the rightmost digit of 72, or 2, and `getCheck` returns the integer value 2.

Two helper methods, `getNumberOfDigits` and `getDigit`, have been provided.

- `getNumberOfDigits` returns the number of digits in its `int` parameter.
- `getDigit` returns the `n`th digit of its `int` parameter.

The following are examples of the use of `getNumberOfDigits` and `getDigit`.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNumberOfDigits(283415)</td>
<td>6</td>
<td>The number 283415 has 6 digits.</td>
</tr>
<tr>
<td>getDigit(283415, 1)</td>
<td>2</td>
<td>The first digit of 283415 is 2.</td>
</tr>
<tr>
<td>getDigit(283415, 5)</td>
<td>1</td>
<td>The fifth digit of 283415 is 1.</td>
</tr>
</tbody>
</table>

Complete the `getCheck` method below. You must use `getNumberOfDigits` and `getDigit` appropriately to receive full credit.

```java
/**
 * Returns the check digit for `num`, as described in part (a).
 * Precondition: The number of digits in `num` is between one and six,
 * inclusive.
 * `num` >= 0
 */
public static int getCheck(int num)
```
(b) Write the isValid method. The method returns true if its parameter numWithCheckDigit, which represents a number containing a check digit, is valid, and false otherwise. The check digit is always the rightmost digit of numWithCheckDigit.

The following table shows some examples of the use of isValid.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCheck(159)</td>
<td>2</td>
<td>The check digit for 159 is 2.</td>
</tr>
<tr>
<td>isValid(1592)</td>
<td>true</td>
<td>The number 1592 is a valid combination of a number (159) and its check digit (2).</td>
</tr>
<tr>
<td>isValid(1593)</td>
<td>false</td>
<td>The number 1593 is not a valid combination of a number (159) and its check digit (3) because 2 is the check digit for 159.</td>
</tr>
</tbody>
</table>

Complete method isValid below. Assume that getCheck works as specified, regardless of what you wrote in part (a). You must use getCheck appropriately to receive full credit.

```java
/**
 * Returns true if numWithCheckDigit is valid, or false otherwise, as described in part (b).
 * Precondition: The number of digits in numWithCheckDigit is between two and seven, inclusive.
 */
public static boolean isValid(int numWithCheckDigit)
```
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.
Scoring Guidelines for Question 1: Methods and Control Structures  

9 points


Canonical solution

(a) public static int getCheck(int num)
{
    int sum = 0;
    for (int i = 1; i <= getNumberOfDigits(num); i++)
    {
        sum += (8 - i) * getDigit(num, i);
    }
    return sum % 10;
}

(b) public static boolean isValid(int numWithCheckDigit)
{
    int check = numWithCheckDigit % 10;
    int num = numWithCheckDigit / 10;
    int newCheck = getCheck(num);
    if (check == newCheck)
    {
        return true;
    }
    else
    {
        return false;
    }
}
## (a) getCheck

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>1 point</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Calls <code>getNumberOfDigits</code> and <code>getDigit</code></td>
<td></td>
<td>3.A MOD-1.H</td>
</tr>
<tr>
<td><strong>2</strong> Calculates one partial sum</td>
<td>Responses earn the point if they...</td>
<td>1 point 3.C CON-1.A</td>
</tr>
<tr>
<td></td>
<td>• Calculate the sum inside or outside the context of a loop.</td>
<td></td>
</tr>
<tr>
<td><strong>3</strong> Calculates all partial sums <em>(in the context of a loop, no bounds errors)</em></td>
<td>Responses earn the point if they...</td>
<td>1 point 3.C CON-2.E</td>
</tr>
<tr>
<td></td>
<td>• Use either a <code>for</code> or <code>while</code> loop.</td>
<td></td>
</tr>
<tr>
<td><strong>4</strong> Returns the last digit of the calculated sum as the check digit</td>
<td>Responses still earn the point even if they...</td>
<td>1 point 3.C CON-1.A</td>
</tr>
<tr>
<td></td>
<td>• Calculate the sum incorrectly.</td>
<td></td>
</tr>
</tbody>
</table>

**Total for part (a)** 4 points

## (b) isValid

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>1 point</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5</strong> Obtains check digit of <code>numWithCheckDigit</code></td>
<td></td>
<td>3.C CON-1.A</td>
</tr>
<tr>
<td><strong>6</strong> Obtains number remaining in <code>numWithCheckDigit</code> after check digit removed</td>
<td></td>
<td>1 point 3.C CON-1.A</td>
</tr>
<tr>
<td><strong>7</strong> Calls <code>getCheck</code> on number without check digit</td>
<td></td>
<td>3.A MOD-1.H</td>
</tr>
<tr>
<td><strong>8</strong> Compares check digit of <code>numWithCheckDigit</code> and return value from <code>getCheck</code></td>
<td>Responses still earn the point even if they...</td>
<td>1 point 3.C CON-1.E</td>
</tr>
<tr>
<td></td>
<td>• Calculated check digit incorrectly; or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Calculated number without check digit incorrectly.</td>
<td></td>
</tr>
<tr>
<td><strong>9</strong> Returns <code>true</code> or <code>false</code> depending on the result of the previous comparison</td>
<td>Responses still earn the point even if they...</td>
<td>1 point 3.C CON-2.A</td>
</tr>
<tr>
<td></td>
<td>• Do not use an <code>if</code> statement</td>
<td></td>
</tr>
</tbody>
</table>

**Total for part (b)** 5 points

**Question specific penalties**

None

**Total for question 1** 9 points
Question 3: Array/ArrayList

The Gizmo class represents gadgets that people purchase. Some Gizmo objects are electronic and others are not. A partial definition of the Gizmo class is shown below.

```java
public class Gizmo {
    /** Returns the name of the manufacturer of this Gizmo. */
    public String getMaker() {
        /* implementation not shown */
    }

    /** Returns true if this Gizmo is electronic, and false otherwise. */
    public boolean isElectronic() {
        /* implementation not shown */
    }

    /** Returns true if this Gizmo is equivalent to the Gizmo object represented by the parameter, and false otherwise. */
    public boolean equals(Object other) {
        /* implementation not shown */
    }

    // There may be instance variables, constructors, and methods not shown.
}
```

The OnlinePurchaseManager class manages a sequence of Gizmo objects that an individual has purchased from an online vendor. You will write two methods of the OnlinePurchaseManager class. A partial definition of the OnlinePurchaseManager class is shown below.

```java
public class OnlinePurchaseManager {
    /** An ArrayList of purchased Gizmo objects, instantiated in the constructor. */
    private ArrayList<Gizmo> purchases;

    /** Returns the number of purchased Gizmo objects that are electronic whose manufacturer is maker, as described in part (a). */
    public int countElectronicsByMaker(String maker) {
        /* to be implemented in part (a) */
    }
}
```
/** Returns true if any pair of adjacent purchased Gizmo objects are equivalent, and false otherwise, as described in part (b). */

public boolean hasAdjacentEqualPair()
{
    /* to be implemented in part (b) */
}

// There may be instance variables, constructors, and methods not shown.

(a) Write the countElectronicsByMaker method. The method examines the ArrayList instance variable purchases to determine how many Gizmo objects purchased are electronic and are manufactured by maker.

Assume that the OnlinePurchaseManager object opm has been declared and initialized so that the ArrayList purchases contains Gizmo objects as represented in the following table.

<table>
<thead>
<tr>
<th>Index in purchases</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value returned by method call isElectronic()</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Value returned by method call getMaker()</td>
<td>&quot;ABC&quot;</td>
<td>&quot;ABC&quot;</td>
<td>&quot;XYZ&quot;</td>
<td>&quot;lmnop&quot;</td>
<td>&quot;ABC&quot;</td>
<td>&quot;ABC&quot;</td>
</tr>
</tbody>
</table>

The following table shows the value returned by some calls to countElectronicsByMaker.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>opm.countElectronicsByMaker(&quot;ABC&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>opm.countElectronicsByMaker(&quot;lmnop&quot;)</td>
<td>0</td>
</tr>
<tr>
<td>opm.countElectronicsByMaker(&quot;XYZ&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>opm.countElectronicsByMaker(&quot;QRP&quot;)</td>
<td>0</td>
</tr>
</tbody>
</table>

Complete method countElectronicsByMaker below.

/** Returns the number of purchased Gizmo objects that are electronic and whose manufacturer is maker, as described in part (a). */

public int countElectronicsByMaker(String maker)
{

(b) When purchasing items online, users occasionally purchase two identical items in rapid succession without intending to do so (e.g., by clicking a purchase button twice). A vendor may want to check a user's purchase history to detect such occurrences and request confirmation.

Write the hasAdjacentEqualPair method. The method detects whether two adjacent Gizmo objects in purchases are equivalent, using the equals method of the Gizmo class. If an adjacent equivalent pair is found, the hasAdjacentEqualPair method returns true. If no such pair is found, or if purchases has fewer than two elements, the method returns false.

Complete method hasAdjacentEqualPair below.

/** Returns true if any pair of adjacent purchased Gizmo objects are equivalent, and false otherwise, as described in part (b). */

public boolean hasAdjacentEqualPair()
Scoring Guidelines for Question 3: Array/ArrayList  


Canonical solution

(a)  public int countElectronicsByMaker(String maker)
   {
       int result = 0;
       for (Gizmo g : purchases)
       {
           if (g.getMaker().equals(maker) && g.isElectronic())
           {
               result++;
           }
       }
       return result;
   }

(b)  public boolean hasAdjacentEqualPair()
   {
       Gizmo g1 = purchases.get(0);
       for (int pos = 1; pos < purchases.size(); pos++)
       {
           Gizmo g2 = purchases.get(pos);
           if (g1.equals(g2))
           {
               return true;
           }
           g1 = g2;
       }
       return false;
   }

4 points

5 points
(a) **countElectronicsByMaker**

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Accesses all elements of <em>purchases</em> (no bounds errors)</td>
<td>Responses earn the point if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>• Use a for, enhanced for, or while loop.</td>
<td></td>
</tr>
<tr>
<td>2. Calls <code>isElectronic</code> to test whether or not a <em>Gizmo</em> object is electronic and <code>getMaker</code> to get the name of the manufacturer of a <em>Gizmo</em> object</td>
<td>Responses earn the point if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>• Call methods on any <em>Gizmo</em> object, regardless of whether this is in the context of a loop.</td>
<td></td>
</tr>
<tr>
<td>3. Tests whether or not the maker of a <em>Gizmo</em> object matches <code>maker</code>, using an appropriate <code>String</code> comparison at least once, in the context of a loop</td>
<td>Responses still earn the point even if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>• Do not return the number of elements that are electronic and made by <code>maker</code></td>
<td></td>
</tr>
</tbody>
</table>

**Total for part (a)** 4 points

(b) **hasAdjacentEqualPair**

<table>
<thead>
<tr>
<th>Scoring Criteria</th>
<th>Decision Rules</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Determines whether the <em>ArrayList</em> contains at least two elements</td>
<td>Responses earn the point if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Accesses all necessary elements of <em>purchases</em> (no bounds error)</td>
<td>Responses still earn the point even if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>• Do not access adjacent elements in the context of a loop</td>
<td></td>
</tr>
<tr>
<td>7. Accesses two adjacent elements of the <em>ArrayList</em> in the context of a comparison</td>
<td>Responses still earn the point even if they...</td>
<td>1 point</td>
</tr>
<tr>
<td></td>
<td>• Use a for or while loop to access the pairs of elements.</td>
<td></td>
</tr>
<tr>
<td>8. Compares two distinct elements of the <em>ArrayList</em> for equivalence</td>
<td></td>
<td>1 point</td>
</tr>
<tr>
<td>9. Returns <code>true</code> if at least one equivalent pair is found, and <code>false</code> otherwise or if the <em>ArrayList</em> has fewer than two elements</td>
<td>Responses earn the point if they...</td>
<td>1 point</td>
</tr>
</tbody>
</table>

**Total for part (b)** 5 points

**Question specific penalties**

None

**Total for question 3** 9 points