

Answer Key and Question Alignment to Course Framework

Multiple-Choice Question	Answer	Skill	Learning Objective	Unit
1	C	2.A	CON-1.A	1
2	A	5.A	VAR-1.E.b	2
3	D	1.C	CON-1.D	2
4	E	4.A	CON-2.A	3
5	B	5.A	CON-1.G	3
6	A	2.B	CON-2.C	4
7	C	1.C	MOD-2.D	5
8	A	4.B	VAR-2.B	6
9	A	1.B	VAR-2.C	6
10	D	2.C	VAR-2.D	7
11	E	5.B	VAR-2.E	7
12	D	2.B	VAR-2.G.a	8
13	B	2.B	VAR-2.G.b	8
14	D	2.C	MOD-3.B	9
15	A	4.C	CON-2.O	10

Free-Response Question	Question Type	Skill	Learning Objective	Unit
1	Methods and Control Structures	3.A, 3.C	MOD-1.H, CON-1.A, CON-1.E, CON-2.A CON-2.E	1,2,3,4,5
3	Array / ArrayList	3.A, 3.C, 3.D	MOD-1.G, CON-1.B, CON-1.H, CON-2.C, CON-2.J.b, VAR-1.E.b, VAR-2.D, VAR-2.E	1,2,3,4,5,7

The scoring information for the questions within this course and exam description, along with further exam resources, can be found on the [AP Computer Science A Exam Page](#) on AP Central.



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.

```
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 nthdigit 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`.

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

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

```
/** 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`.

Method Call	Return Value	Explanation
<code>getCheck(159)</code>	2	The check digit for 159 is 2.
<code>isValid(1592)</code>	<code>true</code>	The number 1592 is a valid combination of a number (159) and its check digit (2).
<code>isValid(1593)</code>	<code>false</code>	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.

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.

```
/** 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)
```

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.

Learning Objectives: MOD-1.H CON-1.A CON-1.E CON-2.A CON-2.E

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;
}
```

4 points

(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;
    }
}
```

5 points

(a) getCheck

Scoring Criteria	Decision Rules	
1 Calls getNumberOfDigits and getDigit		1 point 3.A MOD-1.H
2 Calculates one partial sum	Responses earn the point if they... <ul style="list-style-type: none">Calculate the sum inside or outside the context of a loop.	1 point 3.C CON-1.A
3 Calculates all partial sums (<i>in the context of a loop, no bounds errors</i>)	Responses earn the point if they... <ul style="list-style-type: none">Use either a for or while loop.	1 point 3.C CON-2.E
4 Returns the last digit of the calculated sum as the check digit	Responses still earn the point even if they... <ul style="list-style-type: none">Calculate the sum incorrectly.	1 point 3.C CON-1.A
Total for part (a)		4 points

(b) isValid

Scoring Criteria	Decision Rules	
5 Obtains check digit of numWithCheckDigit		1 point 3.C CON-1.A
6 Obtains number remaining in numWithCheckDigit after check digit removed		1 point 3.C CON-1.A
7 Calls getCheck on number without check digit		1 point 3.A MOD-1.H
8 Compares check digit of numWithCheckDigit and return value from getCheck	Responses still earn the point even if they... <ul style="list-style-type: none">Calculated check digit incorrectly; orCalculated number without check digit incorrectly.	1 point 3.C CON-1.E
9 Returns true or false depending on the result of the previous comparison	Responses still earn the point even if they... <ul style="list-style-type: none">Do not use an if statement	1 point 3.C CON-2.A
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.

```
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.

```
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.

Index in purchases	0	1	2	3	4	5
Value returned by method call <code>isElectronic()</code>	true	false	true	false	true	false
Value returned by method call <code>getMaker()</code>	"ABC"	"ABC"	"XYZ"	"lmnop"	"ABC"	"ABC"

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

Method Call	Return Value
<code>opm.countElectronicsByMaker("ABC")</code>	2
<code>opm.countElectronicsByMaker("lmnop")</code>	0
<code>opm.countElectronicsByMaker("XYZ")</code>	1
<code>opm.countElectronicsByMaker("QRP")</code>	0

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()

```

Learning Objectives: MOD-1.G CON-1.B CON-1.H CON-2.C CON-2.J.B VAR-1.E.B VAR-2.D VAR 2.E**Canonical solution**

(a) `public int countElectronicsByMarker(String maker)`

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

4 points

(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;
}
```

5 points

(a) countElectronicsByMaker

Scoring Criteria	Decision Rules	
1 Accesses all elements of purchases (no bounds errors)	Responses earn the point if they... <ul style="list-style-type: none">Use a for, enhanced for, or while loop.	1 point 3.D VAR 2.E
2 Calls isElectronic to test whether or not a Gizmo object is electronic and getMaker to get the name of the manufacturer of a Gizmo object	Responses earn the point if they... <ul style="list-style-type: none">Call methods on any Gizmo object, regardless of whether this is in the context of a loop.	1 point 3.A MOD-1.G
3 Tests whether or not the maker of a Gizmo object matches maker, using an appropriate String comparison at least once, in the context of a loop		1 point 3.C VAR-1.E.b
4 Computes the number of elements that are electronic and made by maker	Responses still earn the point even if they... <ul style="list-style-type: none">Do not return the number of elements that are electronic and made by maker	1 point 3.C CON-1.B
Total for part (a)		4 points

(b) hasAdjacentEqualPair

Scoring Criteria	Decision Rules	
5 Determines whether the ArrayList contains at least two elements		1 point 3.D VAR-2.D
6 Accesses all necessary elements of purchases (no bounds error)		1 point 3.D CON-2.J.b
7 Accesses two adjacent elements of the ArrayList in the context of a comparison	Responses still earn the point even if they... <ul style="list-style-type: none">Do not access adjacent elements in the context of a loop.	1 point 3.D VAR 2.E
8 Compares two distinct elements of the ArrayList for equivalence		1 point 3.C CON-1.H
9 Returns true if at least one equivalent pair is found, and false otherwise or if the ArrayList has fewer than two elements	Responses earn the point if they... <ul style="list-style-type: none">Use a for or while loop to access the pairs of elements.	1 point 3.C CON-2.C
Total for part (b)		5 points

Question specific penalties

None

Total for question 3 9 points