

```

public static int mystery2(int n)
{
    int total = 0;
    int x = 1;
    while (x < n)
    {
        total += 5;
        x++;
    }
    return total;
}

```

Which, if any, of the following changes to `mystery2` is required so that the two methods work as intended?

- (A) The variable `total` should be initialized to 1.
- (B) The variable `x` should be initialized to 0.
- (C) The condition in the `while` loop header should be `x < n - 1`.
- (D) The condition in the `while` loop header should be `x <= n`.
- (E) No change is required; the methods, as currently written, return the same values when they are called with the same positive integer parameter `n`.

Section II: Free-Response

The following are examples of the kinds of free-response questions found on the exam. Note that on the actual AP Exam, there will be four free-response questions.

Methods and Control Structures (Free-Response Question 1 on the AP Exam)

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 `nth` 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)

```

Array/ArrayList (Free-Response Question 3 on the AP Exam)

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()
```