Name

NotAnAprilFools'Joke

1. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

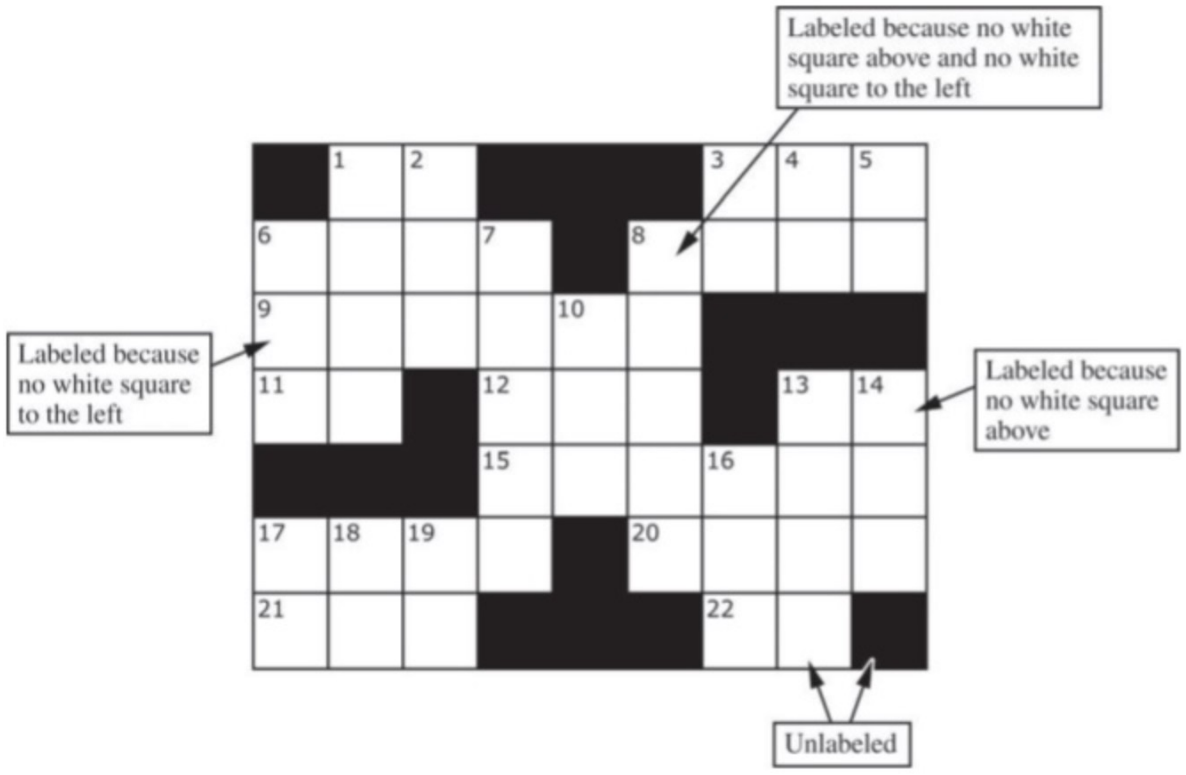
The crossword labeling rule identifies squares to be labeled with a positive number as follows.

A square is labeled with a positive number if and only if

- the square is white and
- the square does not have a white square immediately above it, or it does not have a white square immediately to its left, or both.

The squares identified by these criteria are labeled with consecutive numbers in row-major order, starting at 1.

The following diagram shows a crossword puzzle grid and the labeling of the squares according to the crossword labeling rule.


Labeled because no white square above and no white square to the left
Labeled because no white square to the left
Labeled because no white square above
Unlabeled
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

This question uses two classes, a Square class that represents an individual square in the puzzle and a Crossword class that represents a crossword puzzle grid. A partial declaration of the Square class is shown below.

```
public class Square
{
    /** Constructs one square of a crossword puzzle grid.
     *  Postcondition:
     *    - The square is black if and only if isBlack is true.
     *    - The square has number num.
     */
    public Square(boolean isBlack, int num)
    {  /* implementation not shown */  }

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

A partial declaration of the Crossword class is shown below. You will implement one method and the constructor in the Crossword class.

NotAnAprilFools'Joke

```
public class Crossword
{
    /** Each element is a Square object with a color (black or white) and a number.
     *  puzzle[r][c] represents the square in row r, column c.
     *  There is at least one row in the puzzle.
     */
    private Square[][] puzzle;

    /** Constructs a crossword puzzle grid.
     *  Precondition: There is at least one row in blackSquares.
     *  Postcondition:
     *    - The crossword puzzle grid has the same dimensions as blackSquares.
     *    - The Square object at row r, column c in the crossword puzzle grid is black
     *      if and only if blackSquares[r][c] is true.
     *    - The squares in the puzzle are labeled according to the crossword labeling rule.
     */
    public Crossword(boolean[][] blackSquares)
    {   /* to be implemented in part (b) */   }

    /** Returns true if the square at row r, column c should be labeled with a positive number;
     *          false otherwise.
     *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
     *  Precondition: r and c are valid indexes in blackSquares.
     */
    private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
    {   /* to be implemented in part (a) */   }

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

a. Write the Crossword method toBeLabeled. The method returns true if the square indexed by row r, column c in a crossword puzzle grid should be labeled with a positive number according to the crossword labeling rule; otherwise it returns false. The parameter blackSquares indicates which squares in the crossword puzzle grid are black.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

Complete method toBeLabeled below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

b. Write the Crossword constructor. The constructor should initialize the crossword puzzle grid to have the same dimensions as the parameter blackSquares. Each element of the puzzle grid should be initialized with a reference to a Square object with the appropriate color and number. The number is positive if the square is labeled and 0 if the square is not labeled.

```
Class information for this question

public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

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

Complete the Crossword constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
```

2. Consider the following incomplete method, which is intended to return the number of integers that evenly divide the integer inputVal. Assume that inputVal is greater than 0.

```
public static int numDivisors(int inputVal)
{
  int count = 0;
  for (int k = 1; k <= inputVal; k++)
  {
    if ( /* condition */ )
    {
      count++;
    }
  }
  return count;
}
```

Which of the following can be used to replace / * condition * / so that *numDivisors* will work as intended?

(A) inputVal % k == 0

(B) k % inputVal == 0

(C) inputVal % k != 0

(D) inputVal / k == 0

(E) k / inputVal > 0

3. Consider the following two classes.

```
public class A
{
  public void show()
  {
    System.out.print("A");
  }
}


public class B extends A
{
  public void show()
  {
    System.out.print("B");
  }
}
```

What is printed as a result of executing the following code segment?

```
A obj = new B();
obj.show();
```

(A) A

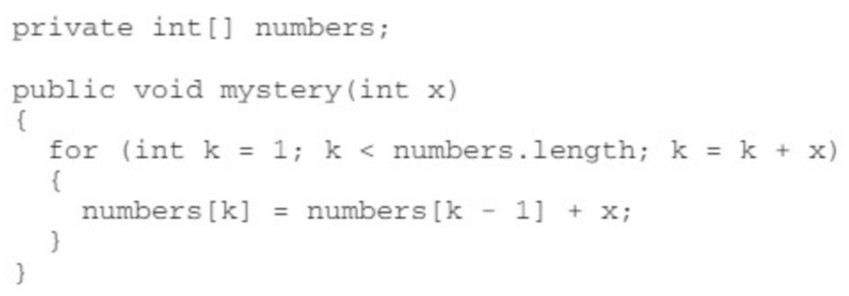
(B) B

(C) AB

(D) BA

(E) The code results in a runtime error.

4. Consider the following instance variable and method.



```
private int[] numbers;

public void mystery(int x)
{
  for (int k = 1; k < numbers.length; k = k + x)
  {
    numbers[k] = numbers[k - 1] + x;
  }
}
```

Assume that numbers has been initialized with the following values.

{17, 34, 21, 42, 15, 69, 48, 25, 39}

Which of the following represents the order of the values in numbers as a result of the call mystery(3)?

(A) {17, 20, 21, 42, 45, 69, 48, 51, 39}

(B) {17, 20, 23, 26, 29, 32, 35, 38, 41}

(C) {17, 37, 21, 42, 18, 69, 48, 28, 39}

(D) {20, 23, 21, 42, 45, 69, 51, 54, 39}

(E) {20, 34, 21, 45, 15, 69, 51, 25, 39}

5. Consider the following method.

```
public static void showMe(int arg)
{
  if (arg < 10)
  {
    showMe(arg + 1);
  }
  else
  {
    System.out.print(arg + " ");
  }
}
```

What will be printed as a result of the call showMe(0) ?

(A) 10

(B) 11

(C) 0 1 2 3 4 5 6 7 8 9

(D) 9 8 7 6 5 4 3 2 1 0

(E) 0 1 2 3 4 5 6 7 8 9 10

6. Consider the following method.

```
/** Precondition: arr.length > 0 */
public static int mystery(int[] arr)
{
  int index = 0;
  int count = 0;
  int m = -1;

  for (int outer = 0; outer < arr.length; outer++)
  {
    count = 0;
    for (int inner = outer + 1; inner < arr.length; inner++)
    {
      if (arr[outer] == arr[inner])
      {
        count++;
      }
    }

    if (count > m)
    {
      index = outer;
      m = count;
    }
  }

  return index;
}
```

Assume that nums has been declared and initialized as an array of integer values. Which of the following best describes the value returned by the call mystery(nums) ?

- (A) The maximum value that occurs in nums
- (B) An index of the maximum value that occurs in nums
- (C) The number of times that the maximum value occurs in nums
- (D) A value that occurs most often in nums
- (E) An index of a value that occurs most often in nums

7. Consider the following two methods, which appear within a single class.

```
public static void changeIt(int[] arr, int val, String word)
{
  arr = new int[5];
  val = 0;
  word = word.substring(0, 5);

  for (int k = 0; k < arr.length; k++)
  {
    arr[k] = 0;
  }
}


public static void start()
{
  int[] nums = {1, 2, 3, 4, 5};
  int value = 6;
  String name = "blackboard";

  changeIt(nums, value, name);

  for (int k = 0; k < nums.length; k++)
  {
    System.out.print(nums[k] + " ");
  }

  System.out.print(value + " ");
  System.out.print(name);
}
```

What is printed as a result of the call start() ?

(A) 0 0 0 0 0 0 black

(B) 0 0 0 0 0 6 blackboard

(C) 1 2 3 4 5 6 black

(D) 1 2 3 4 5 0 black

(E) 1 2 3 4 5 6 blackboard

---

8. Consider the following class.

```
public class SomeMethods

{
public void one(int first)

{  / * implementation not shown * /  }

public void one(int first, int second)

{   / * implementation not shown * /  }

public void one(int first, String second)

{  / * implementation not shown * /   }

}
```

Which of the following methods can be added to the SomeMethods class without causing a compile-time error?

I. public void one(int value)
   {   / * implementation not shown * /   }

II. public void one (String first, int second

   {   / * implementation not shown * /   }

III. public void one (int first, int second, int third)

   {   / * implementation not shown * /   }

(A) I only

(B) I and II only

(C) I and III only

(D) II and III only

(E) I, II, and III

---

9. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

(a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write an interface named NumberGroup that represents a group of integers. The interface should have a single contains method that determines if a given integer is in the group. For example, if group1 is of type NumberGroup, and it contains only the two numbers -5 and 3, then group1.contains(-5) would return true, and group1.contains(2) would return false.
Write the complete NumberGroup interface. It must have exactly one method.

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.
Write the Range class, which is a NumberGroup. The Range class represents the group of int values that range from a given minimum value up through a given maximum value, inclusive. For example,the declaration

NumberGroup range1 = new Range(-3, 2);

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete Range class. Include all necessary instance variables and methods as well as a constructor that takes two int parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

(c) The MultipleGroups class (not shown) represents a collection of NumberGroup objects and isa NumberGroup. The MultipleGroups class stores the number groups in the instance variable groupList (shown below), which is initialized in the constructor.

private List<NumberGroup> groupList;

Write the MultipleGroups method contains. The method takes an integer and returns true if and only if the integer is contained in one or more of the number groups in groupList.

For example, suppose multiple1 has been declared as an instance of MultipleGroups and consists of the three ranges created by the calls new Range(5, 8), new Range(10, 12), and new Range(1, 6). The following table shows the results of several calls to contains.

Call	Result
`multiple1.contains(2)`	`true`
`multiple1.contains(9)`	`false`
`multiple1.contains(6)`	`true`

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *          false otherwise.
 */
public boolean contains(int num)
```

10. Consider the following code segment.

```
List<String> animals = new ArrayList<String>();

animals.add("dog");
animals.add("cat");
animals.add("snake");
animals.set(2, "lizard");
animals.add(1, "fish");
animals.remove(3);
System.out.println(animals);
```

What is printed as a result of executing the code segment?

(A) [dog, fish, cat]

(B) [dog, fish, lizard]

(C) [dog, lizard, fish]

(D) [fish, dog, cat]

(E) The code throws an ArrayIndexOutOfBoundsException exception.

11. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters and has a length known to the player. A guess contains only capital letters and has the same length as the hidden word.

After a guess is made, the player is given a hint that is based on a comparison between the hidden word and the guess. Each position in the hint contains a character that corresponds to the letter in the same position in the guess. The following rules determine the characters that appear in the hint.

If the letter in the guess is ...	the corresponding character in the hint is
also in the same position in the hidden word,	the matching letter
also in the hidden word, but in a different position,	"+"
not in the hidden word,	"*"

The HiddenWord class will be used to represent the hidden word in the game. The hidden word is passed to the constructor. The class contains a method, getHint, that takes a guess and produces a hint.

For example, suppose the variable puzzle is declared as follows.

HiddenWord puzzle = new HiddenWord("HARPS");

The following table shows several guesses and the hints that would be produced.

Call to `getHint`	String returned
`puzzle.getHint("AAAAA")`	`"+A+++"`
`puzzle.getHint("HELLO")`	`"H****"`
`puzzle.getHint("HEART")`	`"H*++*"`
`puzzle.getHint("HARMS")`	`"HAR*S"`
`puzzle.getHint("HARPS")`	`"HARPS"`

Write the complete HiddenWord class, including any necessary instance variables, its constructor, and the method, getHint, described above. You may assume that the length of the guess is the same as the length of the hidden word.

---

12. Consider the following method.

```
/** Precondition: 0 < numVals <= nums.length */
public static int mystery(int[] nums, int v, int numVals)
{
  int k = 0;

  if (v == nums[numVals - 1])
  {
    k = 1;
  }

  if (numVals == 1)
  {
    return k;
  }
  else
  {
    return k + mystery(nums, v, numVals - 1);
  }
}
```

Which of the following best describes what the call mystery(numbers, val, numbers.length) does? You may assume that variables numbers and val have been declared and initialized.

(A) Returns 1 if the last element in numbers is equal to val; otherwise, returns 0

(B) Returns the index of the last element in numbers that is equal to val

(C) Returns the number of elements in numbers that are equal to val

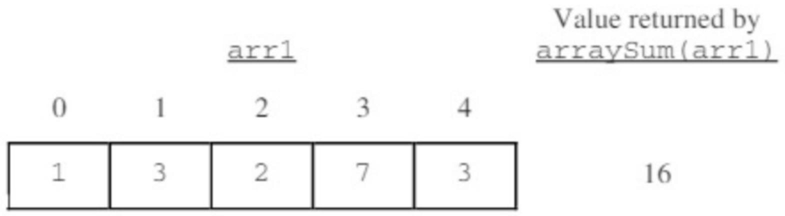
(D) Returns the number of elements in numbers that are not equal to val

(E) Returns the maximum number of adjacent elements that are not equal to val

---

13. This question involves reasoning about one-dimensional and two-dimensional arrays of integers. You will write three static methods, all of which are in a single enclosing class, named DiverseArray (not shown). The first method returns the sum of the values of a one-dimensional array; the second method returns an array that represents the sums of the rows of a two-dimensional array; and the third method analyzes row sums.

(a) Write a static method arraySum that calculates and returns the sum of the entries in a specified one-dimensional array. The following example shows an array arr1 and the value returned by a call to arraySum.



Complete method arraySum below.

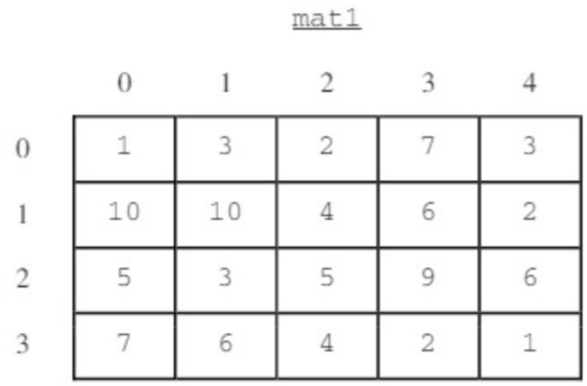
/ * * Returns the sum of the entries in the one-dimensional array arr.

* /

public static int arraySum (int [ ] arr)

(b) Write a static method rowSums that calculates the sums of each of the rows in a given two-dimensional array and returns these sums in a one-dimensional array. The method has one parameter, a two-dimensional array arr2D of int values. The array is in row-major order: arr2D [r] [c] is the entry at row r and column c. The method returns a one-dimensional array with one entry for each row of arr2D such that each entry is the sum of the corresponding row in arr2D. As a reminder, each row of a two-dimensional array is a one-dimensional array.

For example, if mat1 is the array represented by the following table, the call rowSums(mat1) returns the array {16, 32, 28, 20}.



mat1

	0	1	2	3	4
0	1	3	2	7	3
1	10	10	4	6	2
2	5	3	5	9	6
3	7	6	4	2	1

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

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

Complete method rowSums below.

/ * * Returns a one-dimensional array in which the entry at index k is the sum of

*   the entries of row k of the two-dimensional array arr2D.

* /

public static int [ ] rowSums(int [ ] [ ] arr2D)

(c) A two-dimensional array is diverse if no two of its rows have entries that sum to the same value. In the following examples, the array mat1 is diverse because each row sum is different, but the array mat2 is not diverse because the first and last rows have the same sum.

mat1

	0	1	2	3	4	Row sums
0	1	3	2	7	3	16
1	10	10	4	6	2	32
2	5	3	5	9	6	28
3	7	6	4	2	1	20

mat2

	0	1	2	3	4	Row sums
0	1	1	5	3	4	14
1	12	7	6	1	9	35
2	8	11	10	2	5	36
3	3	2	3	0	6	14

Write a static method isDiverse that determines whether or not a given two-dimensional array is diverse. The method has one parameter: a two-dimensional array arr2D of int values. The method should return true if all the row sums in the given array are unique; otherwise, it should return false. In the arrays shown above, the call isDiverse (mat1) returns true and the call isDiverse(mat2) returns false.

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

Assume that arraySum and rowSums work as specified, regardless of what you wrote in parts (a) and(b). You must use rowSums appropriately to receive full credit.
Complete method isDiverse below.

/ * * Returns true if all rows in arr2D have different row sums;
* false otherwise.

* /public static boolean isDiverse(int [ ] [ ] arr2D)

Consider the following instance variable nums and method findLongest with line numbers added for reference. Method findLongest is intended to find the longest consecutive block of the value target occurring in the array nums; however, findLongest does not work as intended.

For example, if the array nums contains the values [7, 10, 10, 15, 15, 15, 15, 10,10, 10, 15, 10, 10], the call findLongest (10) should return 3, the length of the longest consecutive block of 10s.

```
          private int[] nums;

          public int findLongest(int target)
          {
             int lenCount = 0;
             int maxLen = 0;

Line 1:      for (int val : nums)
Line 2:      {
Line 3:        if (val == target)
Line 4:        {
Line 5:          lenCount++;
Line 6:        }
Line 7:        else
Line 8:        {
Line 9:          if (lenCount > maxLen)
Line 10:         {
Line 11:           maxLen = lenCount;
Line 12:         }
Line 13:       }
Line 14:     }
Line 15:     if (lenCount > maxLen)
Line 16:     {
Line 17:       maxLen = lenCount;
Line 18:     }
Line 19:     return maxLen;
          }
```

14. Which of the following changes should be made so that method findLongest will work as intended?

(A) Insert the statement lenCount = 0;
between lines 2 and 3.

(B) Insert the statement lenCount = 0;
between lines 8 and 9.

(C) Insert the statement lenCount = 0;
between lines 10 and 11.

(D) Insert the statement lenCount = 0;
between lines 11 and 12.

(E) Insert the statement lenCount = 0;
between lines 12 and 13.

Directions: Select the choice that best fits each statement. The following question(s) refer to the following incomplete class declaration.

```
public class TimeRecord
{
  private int hours;
  private int minutes; // 0 ≤ minutes < 60
  /** Constructs a TimeRecord object.
   *  @param h  the number of hours
   *            Precondition: h ≥ 0
   *  @param m  the number of minutes
   *            Precondition: 0 ≤ m < 60
   */
  public TimeRecord(int h, int m)
  {
    hours = h;
    minutes = m;
  }

  /** @return the number of hours
   */
  public int getHours()
  { /* implementation not shown */ }

  /** @return the number of minutes
   *  Postcondition: 0 ≤ minutes < 60
   */
  public int getMinutes()
  { /* implementation not shown */ }

  /** Adds h hours and m minutes to this TimeRecord.
   *  @param h  the number of hours
   *            Precondition: h ≥ 0
   *  @param m  the number of minutes
   *            Precondition: m ≥ 0
   */
  public void advance(int h, int m)
  {
    hours = hours + h;
    minutes = minutes + m;
    /* missing code */
  }
  // Other methods not shown
```

15. Consider the following declaration that appears in a class other than TimeRecord.

TimeRecord [ ] timeCards = new TimeRecord [100] ;

Assume that timeCards has been initialized with TimeRecord objects. Consider the following code segment that is intended to compute the total of all the times stored in timeCards.

```
TimeRecord total = new TimeRecord(0,0);
for (int k = 0; k < timeCards.length; k++)
{
   /* missing expression */ ;
}
```

Which of the following can be used to replace / * *missing expression* * / so that the code segment will work as intended?

(A) timeCards [k] .advance ( )

(B) total += timeCards [k] .advance ( )

(C)
```
total.advance(timeCards[k].hours,
              timeCards[k].minutes)
```

(D)
```
total.advance(timeCards[k].getHours(),
              timeCards[k].getMinutes())
```

(E)
```
timeCards[k].advance(timeCards[k].getHours(),
                     timeCards[k].getMinutes())
```

Directions: Select the choice that best fits each statement. The following question(s) refer to the following information

Consider the following instance variable and methods. You may assume that data has been initialized with length > 0. The methods are intended to return the index of an array element equal to target, or -1 if no such element exists.

```
private int[] data;

public int seqSearchRec(int target)
{
  return seqSearchRecHelper(target, data.length - 1);
}


private int seqSearchRecHelper(int target, int last)
{
  // Line 1

  if (data[last] == target)
    return last;
  else
    return seqSearchRecHelper(target, last - 1);
}
```

16. For which of the following test cases will the call seqSearchRec(5) *always* result in an error?

I. data contains only one element.

II. data does not contain the value 5.

III. data contains the value 5 multiple times.

(A) I only

(B) II only

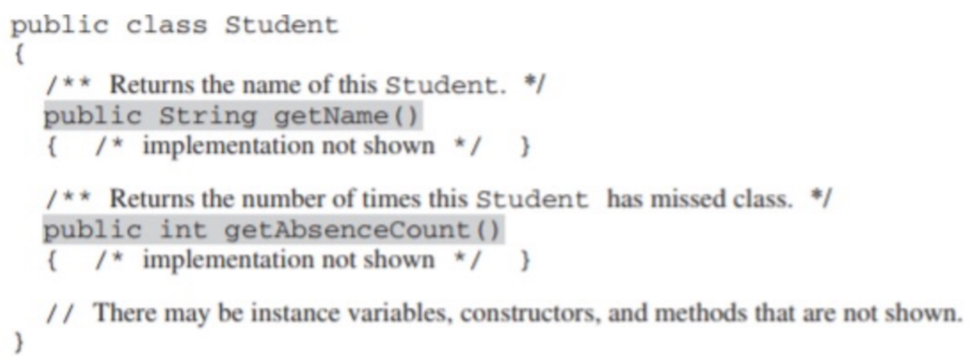
(C) III only

(D) I and II only

(E) I, II, and III

---

17. A student in a school is represented by the following class.



```
public class Student
{
   /** Returns the name of this Student. */
   public String getName()
   {  /* implementation not shown */   }

   /** Returns the number of times this Student has missed class. */
   public int getAbsenceCount()
   {  /* implementation not shown */   }

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

The class SeatingChart, shown below, uses a two-dimensional array to represent the seating arrangement of students in a classroom. The seats in the classroom are in a rectangular arrangement of rows and columns.

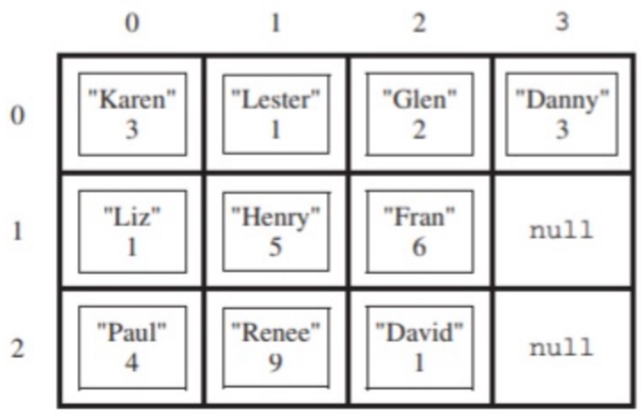
```
public class SeatingChart
{
   /** seats[r][c] represents the Student in row r and column c in the classroom. */
   private Student[][] seats;

   /** Creates a seating chart with the given number of rows and columns from the students in
    *   studentList. Empty seats in the seating chart are represented by null.
    *   @param rows  the number of rows of seats in the classroom
    *   @param cols  the number of columns of seats in the classroom
    *   Precondition: rows > 0; cols > 0;
    *                 rows * cols >= studentList.size()
    *   Postcondition:
    *      - Students appear in the seating chart in the same order as they appear
    *        in studentList, starting at seats[0][0].
    *      - seats is filled column by column from studentList, followed by any
    *        empty seats (represented by null).
    *      - studentList is unchanged.
    */
   public SeatingChart(List<Student> studentList,
                       int rows, int cols)
   {   /* to be implemented in part (a) */   }

   /** Removes students who have more than a given number of absences from the
    *   seating chart, replacing those entries in the seating chart with null
    *   and returns the number of students removed.
    *   @param allowedAbsences  an integer >= 0
    *   @return  number of students removed from seats
    *   Postcondition:
    *     - All students with allowedAbsences or fewer are in their original positions in seats.
    *     - No student in seats has more than allowedAbsences absences.
    *     - Entries without students contain null.
    */
   public int removeAbsentStudents(int allowedAbsences)
   {   /* to be implemented in part (b) */   }

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

a. Write the constructor for the SeatingChart class. The constructor initializes the seats instance variable to a two-dimensional array with the given number of rows and columns. The students in studentList are copied into the seating chart in the order in which they appear in studentList. The students are assigned to consecutive locations in the array seats, starting at seats[0][0] and filling the array column by column. Empty seats in the seating chart are represented by null.

For example, suppose a variable List roster contains references to Student objects in the following order.

"Karen" 3	"Liz" 1	"Paul" 4	"Lester" 1	"Henry" 5	"Renee" 9	"Glen" 2	"Fran" 6	"David" 1	"Danny" 3

A SeatingChart object created with the call new SeatingChart(roster, 3, 4) would have seats initialized with the following values.

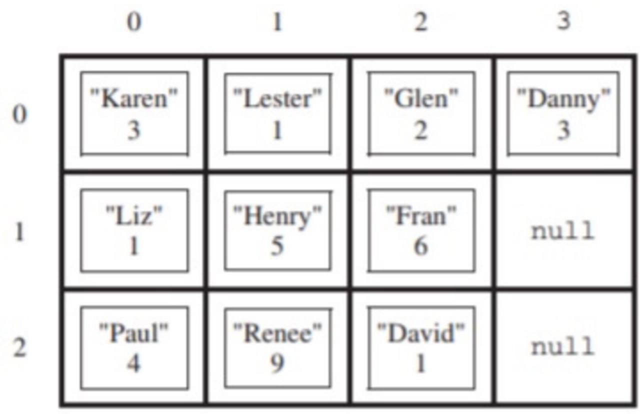


	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

Complete the SeatingChart constructor below.

```
/** Creates a seating chart with the given number of rows and columns from the students in
 *  studentList. Empty seats in the seating chart are represented by null.
 *  @param rows the number of rows of seats in the classroom
 *  @param cols the number of columns of seats in the classroom
 *  Precondition: rows > 0; cols > 0;
 *                rows * cols >= studentList.size()
 *  Postcondition:
 *     - Students appear in the seating chart in the same order as they appear
 *       in studentList, starting at seats[0][0].
 *     - seats is filled column by column from studentList, followed by any
 *       empty seats (represented by null).
 *     - studentList is unchanged.
 */
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
```

b. Write the removeAbsentStudents method, which removes students who have more than a given number of absences from the seating chart and returns the number of students that were removed. When a student is removed from the seating chart, a null is placed in the entry for that student in the array seats. For example, suppose the variable SeatingChart introCS has been created such that the array seats contains the following entries showing both students and their number of absences.



	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

After the call introCS.removeAbsentStudents(4) has executed, the array seats would contain the following values and the method would return the value 3.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	null	null	null
2	"Paul" 4	null	"David" 1	null

Class information repeated from the beginning of the question:

```
public class Student

public String getName()
public int getAbsenceCount()

public class SeatingChart

private Student[][] seats
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
public int removeAbsentStudents(int allowedAbsences)
```

Complete method removeAbsentStudents below.

```
/** Removes students who have more than a given number of absences from the
 *  seating chart, replacing those entries in the seating chart with null
 *  and returns the number of students removed.
 *  @param allowedAbsences an integer >= 0
 *  @return number of students removed from seats
 *  Postcondition:
 *    - All students with allowedAbsences or fewer are in their original positions in seats.
 *    - No student in seats has more than allowedAbsences absences.
 *    - Entries without students contain null.
 */
public int removeAbsentStudents(int allowedAbsences)
```

18. The following sort method correctly sorts the integers in elements into ascending order

```
Line 1:   public static void sort(int[] elements)
Line 2:   {
Line 3:     for (int j = 0; j < elements.length - 1; j++)
Line 4:     {
Line 5.       int index = j;
Line 6:
Line 7:       for (int k = j + 1; k < elements.length; k++)
Line 8:       {
Line 9:         if (elements[k] < elements[index])
Line 10:        {
Line 11:          index = k;
Line 12:        }
Line 13:      }
Line 14:
Line 15:      int temp = elements[j];
Line 16:      elements[j] = elements[index];
Line 17:      elements[index] = temp;
Line 18:    }
Line 19:  }
```

Which of the following changes to the sort method would correctly sort the integers in elements into descending order?

I. Replace line 9 with:

```
Line 9:         if (elements[k] > elements[index])
```

II. Replace lines 15–17 with:

```
Line 15:      int temp = elements[index];
Line 16:      elements[index] = elements[j];
Line 17:      elements[j] = temp;
```

III. Replace line 3 with:

```
Line 3:     for (int j = elements.length − 1; j > 0; j−−)
```

and replace line 7 with:

```
Line 7:       for (int k = 0; k < j; k++)
```

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

---

19. Consider the following method.

```
public static int mystery(int[] arr)
{
  int x = 0;

  for (int k = 0; k < arr.length; k = k + 2)
    x = x + arr[k];

  return x;
}
```

Assume that the array nums has been declared and initialized as follows.

int [ ] nums = { 3,   6,   1,   0,   1,   4,   2};

What value will be returned as a result of the call mystery(nums) ?

(A) 5

(B) 6

(C) 7

(D) 10

20. Consider the following Util class, which contains two methods. The completed sum1D method returns the sum of all the elements of the 1-dimensional array a. The incomplete sum2D method is intended to return the sum of all the elements of the 2-dimensional array m.

```
public class Util
{
   /** Returns the sum of the elements of the 1-dimensional array a */
   public static int sum1D(int[] a)
   {  /* implementation not shown */  }

   /** Returns the sum of the elements of the 2-dimensional array m */
   public static int sum2D(int[][] m)
   {
      int sum = 0;

      /* missing code */

      return sum;
   }
}
```

Assume that sum1D works correctly. Which of the following can replace / * *missing code* * / so that the sum2D method works correctly?

```
  I. for (int k = 0; k < m.length; k++)
     {
        sum += sum1D(m[k]);
     }

 II. for (int[] row : m)
     {
        sum += sum1D(row);
     }

III. for (int[] row : m)
     {
        for (int v : row)
        {
           sum += v;
        }
     }
```

(A) I only

(B) II only

(C) I and II only

(D) II and III only

(E) I, II, and III