Practice Tests

How to Calculate Your (Approximate) AP Computer Science A Score

Multiple-Choice

Number correct (out of 40) = _____ ⟸ Multiple-Choice Score

Free-Response

Question 1 _____ (out of 9)

Question 2 _____ (out of 9)

Question 3 _____ (out of 9)

Question 4 _____ (out of 9)

Total _____ × 1.11 = _____ ⟸ Free-Response Score (Do not round.)

Final Score

_____	+	_____	=	_____
Multiple-Choice Score		Free-Response Score		Final Score (Round to nearest whole number.)

Chart to Convert to AP Grade Computer Science A

Final Score Range	AP Grade[a]

62–80	5
47–61	4
37–46	3
29–36	2
0–28	1

[a]The score range corresponding to each grade varies from exam to exam and is approximate.

Practice Test 1

COMPUTER SCIENCE A
SECTION I

Time—1 hour and 30 minutes
40 Questions

DIRECTIONS: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. Do not spend too much time on any one problem.

NOTES:

- Assume that the classes in the Quick Reference have been imported where needed.
- Assume that variables and methods are declared within the context of an enclosing class.
- Assume that method calls that have no object or class name prefixed, and that are not shown within a complete class definition, appear within the context of an enclosing class.
- Assume that parameters in method calls are not `null` unless otherwise stated.

1. A large Java program was tested extensively, and no errors were found. What can be concluded?
   (A) All of the preconditions in the program are correct.
   (B) All of the postconditions in the program are correct.
   (C) The program may have bugs.
   (D) The program has no bugs.
   (E) Every method in the program may safely be used in other programs.

Questions 2–4 refer to the `Worker` class below.

```
public class Worker
{
    private String name;
    private double hourlyWage;
    private boolean isUnionMember;

    public Worker()
    { /* implementation not shown */ }

    public Worker(String aName, double anHourlyWage, boolean union)
    { /* implementation not shown */ }

    //Accessors getName, getHourlyWage, getUnionStatus are not shown.

    /** Permanently increase hourly wage by amt.
     *  @param amt the amount of wage increase
     */
    public void incrementWage(double amt)
    { /* implementation of incrementWage */ }

    /** Switch value of isUnionMember from true to false and
     *  vice versa.
     */
    public void changeUnionStatus()
    { /* implementation of changeUnionStatus */ }
}
```

2. Refer to the `incrementWage` method. Which of the following is a correct `/* implementation of incrementWage */`?

   (A) `return hourlyWage + amt;`

   (B) `return getHourlyWage() + amt;`

   (C) `hourlyWage += amt;`

   (D) `getHourlyWage() += amt;`

   (E) `hourlyWage = amt;`

3. Consider the method `changeUnionStatus`. Which is a correct `/*` ***implementation of*** `changeUnionStatus */`?

I.
```
if (isUnionMember)
    isUnionMember = false;
else
    isUnionMember = true;
```

II.
```
isUnionMember = !isUnionMember;
```

III.
```
if (isUnionMember)
    isUnionMember = !isUnionMember;
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III

4. A client method `computePay` will return a worker's pay based on the number of hours worked.

```
/** Precondition:  Worker w has worked the given number of hours.
 *  @param w a Worker
 *  @param hours the number of hours worked
 *  @return amount of pay for Worker w
 */
public static double computePay(Worker w, double hours)
{ /* code */ }
```

Which replacement for `/*` ***code*** `*/` is correct?

(A) `return hourlyWage * hours;`
(B) `return getHourlyWage() * hours;`
(C) `return w.getHourlyWage() * hours;`
(D) `return w.hourlyWage * hours;`
(E) `return w.getHourlyWage() * w.hours;`

5. Consider this program segment. You may assume that `wordList` has been declared as `ArrayList<String>`.

```
for (String s : wordList)
    if (s.length() < 4)
        System.out.println("SHORT WORD");
```

What is the maximum number of times that `SHORT WORD` can be printed?

(A) `3`
(B) `4`
(C) `s.length()`
(D) `wordList.size() - 1`
(E) `wordList.size()`

6. Refer to the following method.

```
public static int mystery(int n)
{
    if (n == 1)
        return 3;
    else
        return 3 * mystery(n - 1);
}
```

What value does `mystery(4)` return?

(A) 3
(B) 9
(C) 12
(D) 27
(E) 81

7. Refer to the following declarations.

```
String[] colors = {"red", "green", "black"};
ArrayList<String> colorList = new ArrayList<String>();
```

Which of the following correctly assigns the elements of the `colors` array to `colorList`? The final ordering of colors in `colorList` should be the same as in the `colors` array.

I.
```
for (String col : colors)
    colorList.add(col);
```

II.
```
for (String col : colorList)
    colors.add(col);
```

III.
```
for (int i = colors.length - 1; i >= 0; i--)
    colorList.add(i, colors[i]);
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

8. Often the most efficient computer algorithms use a divide-and-conquer approach, for example, one in which a list is repeatedly split into two pieces until a desired outcome is reached. Which of the following use a divide-and-conquer approach?

I. Merge sort
II. Insertion sort
III. Binary search

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

9. An `Insect` class is to be written, containing the following data fields.

   `age`, which will be initialized to `0` when an `Insect` is constructed.

   `nextAvailableID`, which will be initialized to `0` outside the constructor and incremented each time an `Insect` is constructed.

   `idNum`, which will be initialized to the current value of `nextAvailableID` when an `Insect` is constructed.

   `position`, which will be initialized to the location in a garden where the `Insect` is placed when it is constructed.

   `direction`, which will be initialized to the direction the `Insect` is facing when placed in the garden.

   Which variable in the `Insect` class should be static?

   (A) `age`

   (B) `nextAvailableID`

   (C) `idNum`

   (D) `position`

   (E) `direction`

Questions 10 and 11 refer to the classes `Address` and `Customer` given below.

```
public class Address
{
    private String street;
    private String city;
    private String state;
    private int zipCode;

    public Address(String aStreet, String aCity, String aState,
            int aZipCode)
    { /* implementation not shown */ }

    //Other methods are not shown.
}

public class Customer
{
    private String name;
    private String phone;
    private Address address;
    private int ID;

    public Customer(String aName, String aPhone, Address anAddr,
            int anID)
    { /* implementation not shown */ }

    public Address getAddress()
    { /* implementation not shown */ }

    public String getName()
    { /* implementation not shown */ }

    public String getPhone()
    { /* implementation not shown */ }

    public int getID()
    { /* implementation not shown */ }

    //Other methods are not shown.
}
```

10. Which of the following correctly creates a `Customer` object `c`?

I.
```
Address a = new Address("125 Bismark St", "Pleasantville",
    "NY", 14850);
Customer c = new Customer("Jack Spratt", "747-1674", a, 7008);
```

II.
```
Customer c = new Customer("Jack Spratt", "747-1674",
    "125 Bismark St, Pleasantville, NY 14850", 7008);
```

III.
```
Customer c = new Customer("Jack Spratt", "747-1674",
    new Address("125 Bismark St", "Pleasantville", "NY", 14850),
    7008);
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

11. Consider an `AllCustomers` class that has the following private instance variable.

```
private Customer[] custList;
```

Given the ID number of a particular customer, a method of the class, `locate`, must find the correct `Customer` record and return the name of that customer. Here is the method `locate`:

```
/** Returns the name of the customer with the specified idNum.
 *  Precondition:  custList contains a complete list of Customer objects.
 */
public String locate(int idNum)
{
    for (Customer c : custList)
        if (c.getID() == idNum)
            return c.getName();
    return null;     //idNum not found
}
```

A more efficient algorithm for finding the matching `Customer` object could be used if

(A) `Customer` objects were in alphabetical order by name.

(B) `Customer` objects were sorted by phone number.

(C) `Customer` objects were sorted by ID number.

(D) the `custList` array had fewer elements.

(E) the `Customer` class did not have an `Address` data member.

12. The following shuffling method is used to shuffle an array `arr` of `int` values. The method assumes the existence of a `swap` method, where `swap(arr,i,j)` interchanges the elements `arr[i]` and `arr[j]`.

```
public static void shuffle (int[] arr)
{
    for (int k = arr.length - 1; k > 0; k--)
    {
        int randIndex = (int) (Math.random() * (k + 1));
        swap(arr, k, randIndex);
    }
}
```

Suppose the initial state of `arr` is `1 2 3 4 5`, and when the method is executed the values generated for `randIndex` are `3, 2, 0`, and `1`, in that order. What will be the final state of `arr`?

(A) `5 2 1 3 4`

(B) `1 2 5 3 4`

(C) `5 4 1 3 2`

(D) `4 5 1 3 2`

(E) `2 5 1 3 4`

13. Refer to method `removeWord`.

```
/** Removes all occurrences of word from wordList.
/** Precondition:  wordList is an ArrayList of String objects.
 *  Postcondition: All occurrences of word have been removed
 *                 from wordList.
 */
public static void removeWord(ArrayList<String> wordList,
                                                 String word)
{
    for (int i = 0; i < wordList.size(); i++)
        if ((wordList.get(i)).equals(word))
            wordList.remove(i);
}
```

The method does not always work as intended. Consider the method call

```
removeWord(wordList, "cat");
```

For which of the following lists will this method call fail?

(A) `The cat sat on the mat`

(B) `The cat cat sat on the mat mat`

(C) `The cat sat on the cat`

(D) `cat`

(E) `The cow sat on the mat`

14. A `Clock` class has hours, minutes, and seconds represented by `int` values. It also has each of the following methods: `setTime` to change the time on a `Clock` to the hour, minute, and second specified; `getTime` to access the time; and `toString` to return the time as a `String`. The `Clock` class has a constructor that allows a `Clock` to be created with three `int` parameters for hours, minutes, and seconds. Consider a two-dimensional array of `Clock` values called `allClocks`. A code segment manipulating `allClocks` is as follows.

```
for (Clock[] row : allClocks)
    for (Clock c : row)
        /* more code */
```

Assuming the `Clock` class works as specified, which replacement for `/* more code */` will cause an error?

I. `System.out.print(c);`

II. `c.setTime(0, 0, 0);`

III. `c = new Clock(0, 0, 0);`

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I and II only

15. Consider the following method that will access a square matrix `mat`.

```
/** Precondition:  mat is initialized and is a square matrix.
 */
public static void printSomething(int[][] mat)
{
    for (int r = 0; r < mat.length; r++)
    {
        for (int c=0; c<=r; c++)
            System.out.print(mat[r][c] + " ");
        System.out.println();
    }
}
```

Suppose `mat` is originally

```
0 1 2 3
4 5 6 7
3 2 1 0
7 6 5 4
```

After the method call `printSomething(mat)` the output will be

(A)
```
0 1 2 3
4 5 6 7
3 2 1 0
7 6 5 4
```

(B)
```
0
4 5
3 2 1
7 6 5 4
```

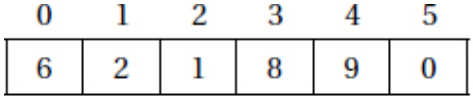
(C)
```
0 1 2 3
4 5 6
3 2
7
```

(D)
```
0
4
3
7
```

(E) There will be no output. An `ArrayIndexOutOfBoundsException` will be thrown.

16. Consider two different ways of storing a set of nonnegative integers in which there are no duplicates.

Method One: Store the integers explicitly in an array in which the number of elements is known. For example, in this method, the set $\{6, 2, 1, 8, 9, 0\}$ can be represented as follows.



6 elements

Method Two: Suppose that the range of the integers is 0 to MAX. Use a boolean array indexed from 0 to MAX. The index values represent the possible values in the set. In other words, each possible integer from 0 to MAX is represented by a different position in the array. A value of true in the array means that the corresponding integer is in the set; a value of false means that the integer is not in the set. For example, using this method for the same set above, $\{6, 2, 1, 8, 9, 0\}$, the representation would be as follows (T = true, F = false).

0	1	2	3	4	5	6	7	8	9	10	…	MAX
T	T	T	F	F	F	T	F	T	T	F	…	F

The following operations are to be performed on the set of integers.

I. Search for a target value in the set.
II. Print all the elements of the set.
III. Return the number of elements in the set.

Which statement is true?

(A) Operation I is more efficient if the set is stored using Method One.
(B) Operation II is more efficient if the set is stored using Method Two.
(C) Operation III is more efficient if the set is stored using Method One.
(D) Operation I is equally efficient for Methods One and Two.
(E) Operation III is equally efficient for Methods One and Two.

17. An algorithm for finding the average of *N* numbers is

$$\text{average} = \frac{\text{sum}}{N}$$

where *N* and sum are both integers. In a program using this algorithm, a programmer forgot to include a test that would check for *N* equal to zero. If *N* is zero, when will the error be detected?

(A) At compile time
(B) At edit time
(C) As soon as the value of *N* is entered
(D) During run time
(E) When an incorrect result is output

18. Consider an array `arr` of 64 distinct `int` values, which are sorted in increasing order. The first element of the array, `arr[0]`, equals `5`, and the last element, `arr[63]`, equals `200`. A binary search algorithm will be used to locate various key values. Which of the following is a true statement?

I. If `5` is the key, it will take exactly `7` iterations of the search loop to locate it.

II. If `2` is the key, it will take exactly `7` iterations of the search loop to determine that `2` is not in `arr`.

III. If `100` is the key, and `100` is equal to `arr[62]`, it will take fewer than `7` iterations of the search loop to locate the key.

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) II and III only

19. Consider method `getCount` below.

```
public static int getCount(String s, String sub)
{
    int count = 0;
    int pos = s.indexOf(sub);
    while (pos >= 0)
    {
        s = s.substring(pos);
        count++;
        pos = s.indexOf(sub);
    }
    return count;
}
```

What will the method call `getCount("a carrot and car", "car")` return?

(A) `0`

(B) `1`

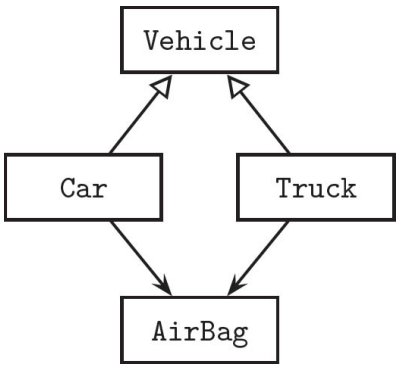
(C) `2`

(D) `3`

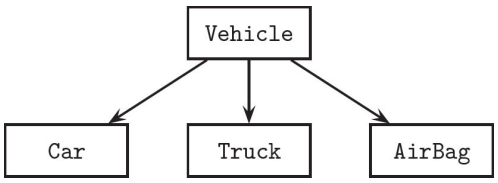
(E) No value returned. The method is in an infinite loop.

20. Consider a program that deals with various components of different vehicles. Which of the following is a reasonable representation of the relationships among some classes that may comprise the program? Note that an open up-arrow denotes an inheritance relationship and a down-arrow denotes a composition relationship.

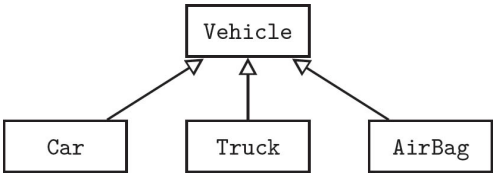
(A)


Vehicle
Car
Truck
AirBag

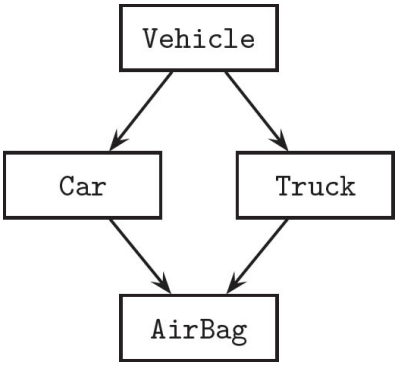
(B)


Vehicle
Car
Truck
AirBag

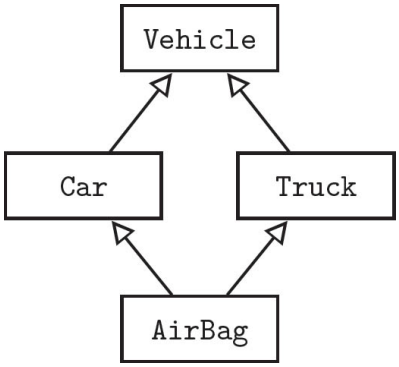
(C)


Vehicle
Car
Truck
AirBag

(D)


Vehicle
Car
Truck
AirBag

(E)


Vehicle
Car
Truck
AirBag

21. Consider the following program segment.

```
/** Precondition:  a[0]...a[n-1] is an initialized array of integers,
 *                 and 0 < n <= a.length.
 */
    int c = 0;
    for (int i = 0; i < n; i++)
        if (a[i] >= 0)
        {
            a[c] = a[i];
            c++;
        }
    n = c;
```

Which is the best postcondition for the segment? (You may assume that `a[0]…a[-1]` represents an empty array.)

(A) `a[0]…a[n-1]` contains no positive integers.

(B) `a[0]…a[n-1]` contains no negative integers.

(C) `a[0]…a[n-1]` contains no nonnegative integers.

(D) `a[0]…a[n-1]` contains no occurrences of zero.

(E) The updated value of `n` is less than or equal to the value of `n` before execution of the segment.

22. If `a`, `b`, and `c` are integers, which of the following conditions is sufficient to guarantee that the expression

```
a < c || a < b && !(a == c)
```

evaluates to true?

(A) `a < c`

(B) `a < b`

(C) `a > b`

(D) `a == b`

(E) `a == c`

23. Airmail Express charges for shipping small packages by integer values of weight. The charges for a weight w in pounds are as follows.

$0 < w \le 2$	\$4.00
$2 < w \le 5$	\$8.00
$5 < w \le 20$	\$15.00

The company does not accept packages that weigh more than 20 pounds. Which of the following represents the best set of data (weights) to test a program that calculates shipping charges?

(A) 0, 2, 5, 20

(B) 1, 4, 16

(C) −1, 1, 2, 3, 5, 16, 20

(D) −1, 0, 1, 2, 3, 5, 16, 20, 22

(E) All integers from −1 through 22

24. Consider the following instance variable and methods in the same class.

```
private int[][] matrix;

/** Precondition:  array.length > 0.
  * @return the largest integer in array
  */
private int max(int[] array)
{  /* implementation not shown */  }

/** @return num1 if num1 >= num2; otherwise return num2
  */
public int max(int num1, int num2)
{  /* implementation not shown */  }
```

Suppose `matrix` has a current value of

```
2 1 4 8
6 0 3 9
5 7 7 6
1 2 3 4
```

What will be returned by the following method call in the same class?

```
max(max(matrix[2]), max(matrix[3]))
```

(A) 9
(B) 8
(C) 7
(D) 4
(E) Compile-time error. No value returned.

Questions 25–26 are based on the following class declaration.

```
public class AutoPart
{
    private String description;
    private int partNum;
    private double price;

    public AutoPart(String desc, int pNum, double aPrice)
    { /* implementation not shown */ }

    public String getDescription()
    { return description; }

    public int getPartNum()
    { return partNum; }

    public double getPrice()
    { return price; }

    //Other methods are not shown.
    //There is no compareTo method.
}
```

25. This question refers to the `findCheapest` method below, which occurs in a class that has an array of `AutoPart` as one of its private data fields.

```
private AutoPart[] allParts;
```

The `findCheapest` method examines an array of `AutoPart` and returns the part number of the `AutoPart` with the lowest price whose description matches the `partDescription` parameter. For example, several of the `AutoPart` elements may have `"headlight"` as their description field. Different headlights will differ in both price and part number. If the `partDescription` parameter is `"headlight"`, then `findCheapest` will return the part number of the cheapest headlight.

```
/** Returns the part number of the cheapest AutoPart
 *  whose description matches partDescription.
 *  Precondition:  allParts contains at least one element whose
 *                 description matches partDescription.
 */
public int findCheapest(String partDescription)
{
    AutoPart part = null;       //AutoPart with lowest price so far
    double min = LARGE_VALUE;   //larger than any valid price
    for (AutoPart p : allParts)
    {
        /* more code */
    }
    return part.getPartNum();
}
```

Which of the following replacements for `/* more code */` will find the correct part number?

I.
```
if (p.getPrice() < min)
{
    min = p.getPrice();
    part = p;
}
```

II.
```
if (p.getDescription().equals(partDescription))
    if (p.getPrice() < min)
    {
        min = p.getPrice();
        part = p;
    }
```

III.
```
if (p.getDescription().equals(partDescription))
    if (p.getPrice() < min)
        return p.getPartNum();
```

(A) I only
(B) II only
(C) III only
(D) I and II only

(E) I and III only

26. Consider the following method.

```
/** Returns the smaller of st1 and st2.
 *  Precondition:  st1 and st2 are distinct String objects.
 */
public static String min(String st1, String st2)
{
    if (st1.compareTo(st2) < 0)
        return st1;
    else
        return st2;
}
```

A method in the same class has these declarations.

```
AutoPart p1 = new AutoPart(< suitable values >);
AutoPart p2 = new AutoPart(< suitable values >);
```

Which of the following statements will cause an error?

I. `System.out.println(min(p1.getDescription(), p2.getDescription()));`

II. `System.out.println(min(p1.toString().getDescription(), p2.toString().getDescription()));`

III. `System.out.println(min(p1, p2));`

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) II and III only

27. This question is based on the following declarations.

```
String strA = "CARROT", strB = "Carrot", strC = "car";
```

Given that all uppercase letters precede all lowercase letters when considering alphabetical order, which is true?

(A) `strA.compareTo(strB) < 0 && strB.compareTo(strC) > 0`

(B) `strC.compareTo(strB) < 0 && strB.compareTo(strA) < 0`

(C) `strB.compareTo(strC) < 0 && strB.compareTo(strA) > 0`

(D) `!(strA.compareTo(strB) == 0) && strB.compareTo(strA) < 0`

(E) `!(strA.compareTo(strB) == 0) && strC.compareTo(strB) < 0`

28. A programmer has a file of names. She is designing a program that sends junk mail letters to everyone on the list. To make the letters sound personal and friendly, she will extract each person's first name from the name string. She plans to create a parallel file of first names only. For example,

fullName	firstName
Ms. Anjali DeSouza	Anjali
Dr. John Roufaiel	John
Mrs. Mathilda Concia	Mathilda

Here is a method intended to extract the first name from a full name string.

```
/** Precondition:
 *   - fullName starts with a title followed by a period.
 *   - A single space separates the title, first name, and last name.
 *  @param fullName a string containing a title, period, blank,
 *  and last name
 *  @return the first name only in fullName
 */
public static String getFirstName(String fullName)
{
    final String BLANK = " ";
    String temp, firstName;

    /* code to extract first name */

    return firstName;
}
```

Which represents correct /* ***code to extract first name*** */?

I.
```
int k = fullName.indexOf(BLANK);
temp = fullName.substring(k + 1);
k = temp.indexOf(BLANK);
firstName = temp.substring(0, k);
```

II.
```
int k = fullName.indexOf(BLANK);
firstName = fullName.substring(k + 1);
k = firstName.indexOf(BLANK);
firstName = firstName.substring(0, k);
```

III.
```
int firstBlank = fullName.indexOf(BLANK);
int secondBlank = fullName.indexOf(BLANK);
firstName = fullName.substring(firstBlank + 1, secondBlank + 1);
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III

Questions 29–31 refer to the `ThreeDigitInteger` and `ThreeDigitCode` classes below.

```
public class ThreeDigitInteger
{
    private int hundredsDigit;
    private int tensDigit;
    private int onesDigit;
    private int value;

    public ThreeDigitInteger(int aValue)
    { /* implementation not shown */ }

    /** Returns the sum of digits for this ThreeDigitInteger. */
    public int digitSum()
    { /* implementation not shown */ }

    /** Returns the sum of the hundreds digit and tens digit. */
    public int twoDigitSum()
    { /* implementation not shown */ }

    //Other methods are not shown.
}

public class ThreeDigitCode extends ThreeDigitInteger
{
    private boolean isValid;

    public ThreeDigitCode(int aValue)
    { /* implementation code */ }

    /** A ThreeDigitCode is valid if and only if the remainder when
     *  the sum of the hundreds and tens digits is divided by 7 equals
     *  the ones digit. Thus 362 is valid while 364 is not.
     *  Returns true if ThreeDigitCode is valid, false otherwise.
     */
    public boolean isValid()
    { /* implementation not shown */ }
}
```

29. Which is a true statement about the classes shown?

(A) The `ThreeDigitInteger` class inherits the `isValid` method from the class `ThreeDigitCode`.

(B) The `ThreeDigitCode` class inherits all of the public accessor methods from the `ThreeDigitInteger` class.

(C) The `ThreeDigitCode` class inherits the constructor from the class `ThreeDigitInteger`.

(D) The `ThreeDigitCode` class can directly access all the private variables of the `ThreeDigitInteger` class.

(E) The `ThreeDigitInteger` class can access the `isValid` instance variable of the `ThreeDigitCode` class.

30. Which is correct `/*` ***implementation code*** `*/` for the `ThreeDigitCode` constructor?

I.
```
super(aValue);
isValid = isValid();
```

II.
```
super(value, valid);
```

III.
```
super(value);
isValid = twoDigitSum() % 7 == onesDigit;
```

(A) I only

(B) II only

(C) III only

(D) I and III only

(E) I, II, and III

31. Refer to these declarations in a client program.

```
ThreeDigitInteger code = new ThreeDigitCode(127);
ThreeDigitInteger num = new ThreeDigitInteger(456);
ThreeDigitCode newCode = new ThreeDigitCode(241);
```

Which of the following subsequent tests will not cause an error?

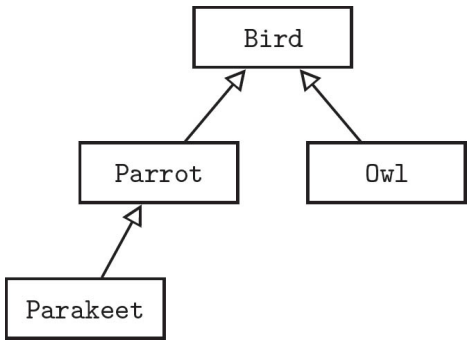
I.
```
if (code.isValid())
    ...
```

II.
```
if (num.isValid())
    ...
```

III.
```
if (newCode.isValid())
    ...
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

32. Consider the following hierarchy of classes.



Assuming that each class has a valid no-argument constructor, which of the following declarations in a client program are correct?

I.
```
Bird b1 = new Parrot();
Bird b2 = new Parakeet();
Bird b3 = new Owl();
```

II.
```
Parakeet p = new Parrot();
Owl o = new Bird();
```

III.
```
Parakeet p = new Bird();
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

33. Consider an array `arr` and a list `list` that is an `ArrayList<String>`. Both `arr` and `list` are initialized with string values. Which of the following code segments correctly appends all the strings in `arr` to the end of `list`?

I.
```
for (String s : arr)
    list.add(s);
```

II.
```
for (String s : arr)
    list.add(list.size(), s);
```

III.
```
for (int i = 0; i < arr.length; i++)
    list.add(arr[i]);
```

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

34. Refer to the `nextIntInRange` method below.

```
/** Returns a random integer in the range low to high, inclusive. */
public int nextIntInRange(int low, int high)
{
    return /* expression */
}
```

Which `/* expression */` will always return a value that satisfies the postcondition?

(A) `(int) (Math.random() * high) + low;`

(B) `(int) (Math.random() * (high - low)) + low;`

(C) `(int) (Math.random() * (high - low + 1)) + low;`

(D) `(int) (Math.random() * (high + low)) + low;`

(E) `(int) (Math.random() * (high + low - 1)) + low;`

35. Consider the following `mergeSort` method and the private instance variable `a` both in the same `Sorter` class.

```
private int[] a;

/** Sorts a[first] to a[last] in increasing order using merge sort. */
public void mergeSort(int first, int last)
{
    if (first != last)
    {
        int mid = (first + last) / 2;
        mergeSort(first, mid);
        mergeSort(mid + 1, last);
        merge(first, mid, last);
    }
}
```

Method `mergeSort` calls method `merge`, which has the following header.

```
/** Merge a[lb] to a[mi] and a[mi+1] to a[ub].
 *  Precondition:  a[lb] to a[mi] and a[mi+1] to a[ub] both
 *                 sorted in increasing order.
 */
private void merge(int lb, int mi, int ub)
```

If the first call to `mergeSort` is `mergeSort(0,3)`, how many *further* calls will there be to `mergeSort` before an array `b[0]…b[3]` is sorted?

(A) 2

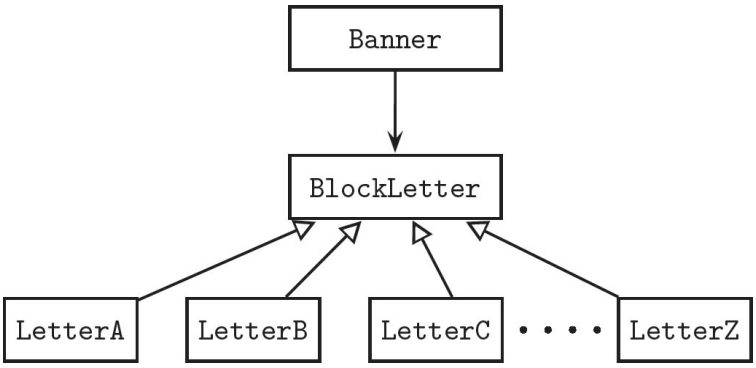
(B) 3

(C) 4
(D) 5
(E) 6

36. A large hospital maintains a list of patients' records in no particular order. To find the record of a given patient, which represents the most efficient method that will work?
    (A) Do a sequential search on the name field of the records.
    (B) Do a binary search on the name field of the records.
    (C) Use insertion sort to sort the records alphabetically by name; then do a sequential search on the name field of the records.
    (D) Use merge sort to sort the records alphabetically by name; then do a sequential search on the name field of the records.
    (E) Use merge sort to sort the records alphabetically by name; then do a binary search on the name field of the records.

Use the following information for Questions 37 and 38.

Here is a diagram that shows the relationship between some of the classes that will be used in a program to draw a banner with block letters.



The diagram shows that the `Banner` class uses `BlockLetter` objects, and that the `BlockLetter` class has 26 subclasses, representing block letters from A to Z.

The `BlockLetter` class has a `draw` method

```
public void draw()
```

Each of the subclasses shown implements the `draw` method in a unique way to draw its particular letter. The `Banner` class gets an array of `BlockLetter` and has a method to draw all the letters in this array.

Here is a partial implementation of the `Banner` class.

```
public class Banner
{
    private BlockLetter[] letters;
    private int numLetters;

    /** Constructor. Gets the letters for the Banner. */
    public Banner()
    {
        numLetters = < some integer read from user input >
        letters = getLetters();
    }
```

```
    /** Returns an array of block letters. */
    public BlockLetter[] getLetters()
    {
        String letter;
        letters = new BlockLetter[numLetters];
        for (int i = 0; i < numLetters; i++)
        {
            < read in capital letter >

            if (letter.equals("A"))
                letters[i] = new LetterA();
            else if (letter.equals("B"))
                letters[i] = new LetterB();
                    ...           //similar code for C through Y
            else
                letters[i] = new LetterZ();
        }
        return letters;
    }

    /** Draw all the letters in the Banner. */
    public void drawLetters()
    {
        for (BlockLetter letter : letters)
            letter.draw();
    }
}
```

37. You are given the information that `Banner` and `BlockLetter` are two classes used in the program. Which of the following can you conclude about the classes?

    I. `BlockLetter` inherits all the methods of `Banner`.

    II. `Banner` contains at least one `BlockLetter` object.

    III. Each of the subclasses `LetterA`, `LetterB`, `LetterC`, ... `LetterZ` has an overridden `draw` method.

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

38. Which is a true statement about method `drawLetters`?
    (A) It is an overloaded method in the `Banner` class.
    (B) It is an overridden method in the `Banner` class.
    (C) It uses polymorphism to draw the correct letters.
    (D) It will cause a logic error because the `draw` method of the `BlockLetter` class is different from the `draw` methods of its subclasses.
    (E) It will cause a run-time error because there is no `draw` method in the `Banner` class.

39. Consider `method1` and `method2` below, which are identical except for the second to last line of code. Each method returns a new matrix based on the input matrix `mat`.

```
public static int[][] method1(int[][] mat)
{
    int numRows = mat.length;
    int numCols = mat[0].length;
    int[][] newMat = new int[numRows][numCols];
    for (int row = 0; row < numRows; row++)
        for (int col = 0; col < numCols; col++)
            newMat[numRows - row -1][col] = mat[row][col];
    return newMat;
}

public static int[][] method2(int[][] mat)
{
    int numRows = mat.length;
    int numCols = mat[0].length;
    int[][] newMat = new int[numRows][numCols];
    for (int row = 0; row < numRows; row++)
        for (int col = 0; col < numCols; col++)
            newMat[row][col] = mat[numRows - row - 1][col];
    return newMat;
}
```

Suppose the same input matrix is used for `method1` and `method2`, and the output for `method1` is `matrix1` while the output for `method2` is `matrix2`. Which is a true statement about `matrix1` and `matrix2`?

(A) `matrix1` is identical to `matrix2`.

(B) The rows of `matrix1` are the columns of `matrix2`.

(C) `matrix1` is a reflection of `matrix2` across a vertical line on the edge of either matrix.

(D) `matrix1` is a reflection of `matrix2` across a horizontal line on the bottom or top edge of either matrix.

(E) The rows of `matrix1` are the rows of `matrix2` in reverse order.

40. Consider an `ArrayList cards` of `Card` objects that needs to be shuffled. The following algorithm is used for shuffling.

Create a temporary `ArrayList<Card>`
Do the following `cards.size()` number of times
- Generate a random integer `r` that can index any card in `cards`
- Remove the card found at position `r` in `cards` and add it to the end of the temporary `ArrayList`

Set `cards` to the temporary `ArrayList`

Here is the method that implements this algorithm.

```
Line 1:  public void shuffle()
Line 2:  {
Line 3:      int size = cards.size();
Line 4:      ArrayList<Card> temp = new ArrayList<Card>();
Line 5:      for (int j = 1; j < size; j++)
Line 6:      {
Line 7:          int index = (int) (Math.random() * size);
Line 8:          temp.add(cards.get(index));
Line 9:      }
Line 10:     cards = temp;
Line 11: }
```

The method does not work as intended. Which of the following changes to `shuffle` would ensure that it works correctly?

I. Replace Line 5 with

```
for (int j = 0; j < size; j++)
```

II. Replace Line 7 with

```
int index = (int) (Math.random() * cards.size());
```

III. Replace Line 8 with

```
temp.add(cards.remove(index));
```

(A) I only
(B) II only

(C) III only
(D) I and III only
(E) I, II, and III

END OF SECTION I

COMPUTER SCIENCE A
SECTION II

Time—1 hour and 30 minutes
4 Questions

DIRECTIONS: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Write your answers in the separate Free-Response booklet provided.

NOTES:

- Assume that the classes in the Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. This question uses a password checker to report whether a given password is weak, medium, or strong. The `PasswordChecker` class is shown below. You will write two methods of the `PasswordChecker` class.

```
public class PasswordChecker
{

    /** Returns the number of digits in s.
     *  Precondition:  s contains at least one character.
     */
    public static int numDigits(String s)
    { /* implementation not shown */ }

    /** Returns the number of letters in s.
     *  Precondition:  s contains at least one character.
     */
    public static int numLetters(String s)
    { /* implementation not shown */ }

    /** Returns the number of characters in s
     *  that are neither letters nor digits.
     *  Precondition:  s contains at least one character.
     */
    public static int numSymbols(String s)
    { /* to be implemented in part (a) */ }

    /** Returns the strength of password p
     *    as described in part (b).
     *  Precondition:  p contains at least one character.
     */
    public static String passwordStrength(String p)
    { /* to be implemented in part (b) */ }

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

(a) Complete the `numSymbols` method, which finds how many characters in `String s` are neither letters nor digits.

Two helper methods, `numDigits` and `numLetters`, have been provided.

- `numDigits` returns the number of digits in its `String` parameter.
- `numLetters` returns the number of letters in its `String` parameter.

The following are some examples showing the use of `numDigits`, `numLetters`, and `numSymbols`.

Method Call	Return Value
`numDigits("R2@n49")`	`3`
`numLetters("R2@n49")`	`2`
`numSymbols("R2@n49")`	`1`
`numDigits("!?!?")`	`0`
`numLetters("!?!?")`	`0`
`numSymbols("!?!?")`	`4`

Complete the `numSymbols` method. You must use `numDigits` and `numLetters` appropriately to receive full credit.

```
/** Returns the number of characters in s
 *  that are neither letters nor digits.
 *  Precondition:  s contains at least one character.
 */
public static int numSymbols(String s)
```

(b) Write the `passwordStrength` method. The method returns one of three `String` values: `"strong"`, `"medium"`, or `"weak"`, depending on the characters of its `String` parameter `p`.

Here are the criteria for each type of password. (Assume that the word "symbol" refers to a character that is neither a digit nor a letter.)

- A strong password is one with at least 8 characters and at least one digit, one letter, and one symbol.

- A medium password has two possibilities:
  - Between 5 and 8 characters (5 inclusive), at least one of which is a symbol.
  - 8 or more characters, but is missing a digit, letter, or symbol, the second condition for being strong.
- A weak password has two possibilities:
  - Fewer than 5 characters.
  - Between 5 and 8 characters (5 inclusive), in which none of the characters is a symbol.

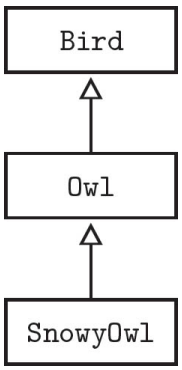
Here are some examples.

Method Call	Return Value
`checkPassword("c@8")`	weak
`checkPassword("c1A2b3")`	weak
`checkPassword("c/A2b3")`	medium
`checkPassword("Two4two?")`	strong
`checkPassword("Hot3dog2019")`	medium

Complete method `passwordStrength`. Assume that `numSymbols` works as specified, regardless of what you wrote in part (a).

```
/** Returns the strength of password p
 *      as described in part (b).
 *  Precondition: p contains at least one character.
 */
public static String passwordStrength(String p)
```

2. In this question, you will write the implementation of a `SnowyOwl` class based on the hierarchy of classes shown below.



The `Owl` class is as follows.

```
public class Owl
{
    private String name;

    public Owl()
    { name = ""; }

    public Owl(String owlName)
    { name = owlName; }

    public String getName()
    { return name; }

    public String getFood()
    { return "furry animals, insects, or small birds"; }
}
```

Here are some features of a `SnowyOwl`.

- It's an `Owl` whose name is always `"Snowy owl"`.
- If the owl is a male, its color is white.
- If it is a female, its color is speckled.
- Food for a `SnowyOwl` depends on what is available. A `SnowyOwl` will randomly eat a hare, a lemming, or a small

bird, where each type of food is equally likely.

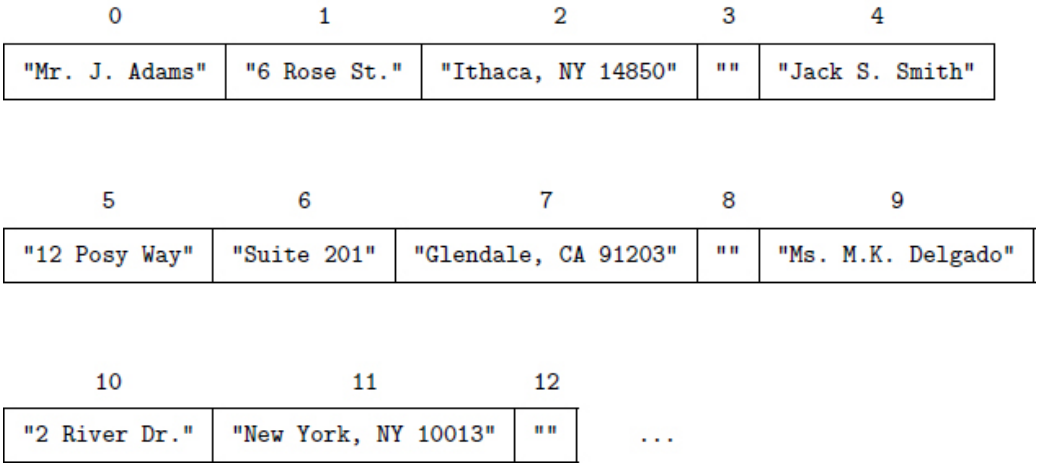
The `SnowyOwl` class should have a private instance variable of type boolean that stores `true` if the owl is male, `false` otherwise. It should also have a constructor and a `getColor` method that returns a string with the snowy owl's color.

Write the complete `SnowyOwl` class. Your implementation should meet all specifications for a `SnowyOwl`.

3. Consider a system for processing names and addresses from a mailing list. A `Recipients` class will be used as part of this system. The lines in the mailing list are stored in an `ArrayList<String>`, a private instance variable in the `Recipients` class. The blank line that separates recipients in the mailing list is stored as the empty string in this list, and the final element in the list is an empty string.

   A portion of the mailing list is shown below, with the corresponding part of the `ArrayList`.

```
Mr. J. Adams
6 Rose St.
Ithaca, NY 14850

Jack S. Smith
12 Posy Way
Suite 201
Glendale, CA 91203

Ms. M.K. Delgado
2 River Dr.
New York, NY 10013

    ...
```



The `Recipients` class that processes these data is shown below.

```
public class Recipients
{
    /** The list of lines in the mailing list */
    private ArrayList<String> lines;

    /** Constructor. Fill lines with mailing list data.
     *  Postcondition:
     *   - Each element in lines is one line of the mailing list.
     *   - Lines appear in the list in the same order
     *     that they appear in the mailing list.
     *   - Blank line separators in the mailing list are stored
     *     as empty strings.
     */
    public Recipients()
    { /* implementation not shown */ }

    /** Returns the city contained in the cityZip string of
     *  an address, as described in part (a).
     */
    public String extractCity(String cityZip)
    { /* to be implemented in part (a) */ }

    /** Returns the address of the recipient with the specified
     *  name, as described in part (b).
     */
    public String getAddress(String name)
    {/* to be implemented in part (b) */}

    //Other methods are not shown.
}
```

(a) Write the `extractCity` method of the `Recipients` class. In the `cityZip` parameter the city is followed by a comma, then one blank space, then two capital letters for a state abbreviation, then a space and 5-digit ZIP code. For example, if `cityZip` is `"Ithaca, NY 14850"`, the method call `extractCity(cityZip)` should return `"Ithaca"`.

Class information for this question

```
public class Recipients

private ArrayList<String> lines
public Recipients()
public String extractCity(String cityZip)
public String getAddress(String name)
```

Complete method `extractCity`.

```
/** Returns the city contained in the cityZip string of
 *  an address, as described in part (a).
 */
public String extractCity(String cityZip)
```

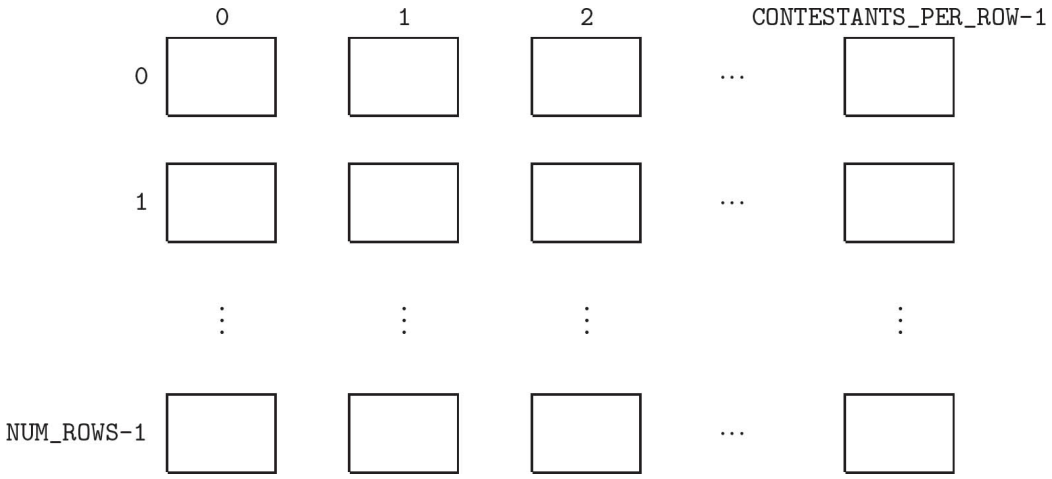
(b) Write the `getAddress` method of the `Recipients` class. This method should return a string that contains only the address of the corresponding `name` parameter. For example, if `name` is `"Jack S. Smith"`, a string containing the three subsequent lines of his address should be returned. This string should contain line breaks in appropriate places, including after the last line of the address. This ensures that the address will have the proper address format when printed by a client class. In the given example of name `"Jack S. Smith"`, the printed version of his address string should look like this:

```
Jack S. Smith
12 Posy Way
Suite 201
Glendale, CA 91203
```

Complete method `getAddress`.

```
/** Returns the address of the recipient with the specified
 *  name, as described in part (b).
 */
public String getAddress(String name)
```

4. A puzzle-solving competition is held in a large hall with a two-dimensional arrangement of contestants. Each rectangle below represents one contestant.



A contestant in the contest can be represented by a `Contestant` class, whose partial implementation follows.

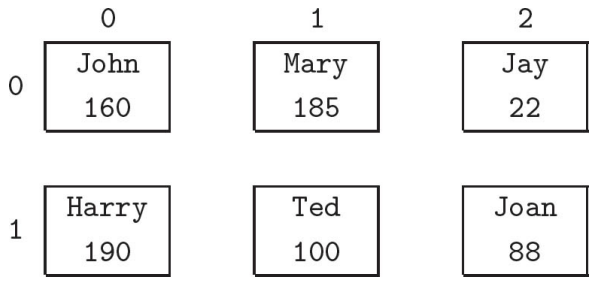
```
public class Contestant
{
    private String name;
    private int score;

    /** Returns the name of this contestant. */
    public String getName()
    { return name; }

    /** Returns the score of this contestant. */
    public int getScore()
    { return score; }

    //Constructor and other methods are not shown.
}
```

In parts (a) and (b) you will write two methods of a `ContestOrganizer` class, whose partial implementation is shown below. A contest organizer keeps track of contestants in a two-dimensional array.

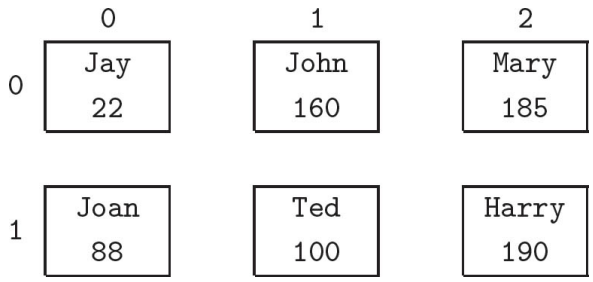
```
public class ContestOrganizer
{
    /** the number of rows of contestants */
    public static final int NUM_ROWS = < some integer >;

    /** the number of columns of contestants */
    public static final int CONTESTANTS_PER_ROW = < some integer >;

    /** The two-dimensional array of contestants */
    private Contestant[][] contestants;

    /** Sorts arr in increasing order by score. */
    private void sort(Contestant[] arr)
    { /* implementation not shown */ }

    /** Sorts each row of contestants into increasing order by score.
     *  Postcondition: Contestant with highest score in row[k] is
     *                 in the rightmost column of row[k], 0<=k<NUM_ROWS.
     */
    public void sortAllRows()
    { /* to be implemented in part(a) */ }

    /** Returns name of contestant with highest score.
     *  Precondition:
     *   - Contestants have not been sorted by score.
     *   - Top score is unique.
     *   - Only one contestant has the highest score.
     */
    public String findWinnerName()
    { /* to be implemented in part(b) */ }
}
```

(a) Write the `ContestOrganizer` method `sortAllRows`. This method should sort the contestants by score in each row, from lowest to highest.

Example: Suppose contestants are as shown below.



Here is what contestants will be after a call to `sortAllRows`.



In writing `sortAllRows`, your method *must* use the `ContestOrganizer` method `sort`. You may assume that `sort` works as specified.

Complete method `sortAllRows`.

```
/** Sorts each row of contestants into increasing order by score.
 *  Postcondition: Contestant with highest score in row[k] is
 *                 in the rightmost column of row[k], 0<=k<NUM_ROWS.
 */
public void sortAllRows()
```

(b) Write the `Contestant` method `findWinnerName`, which returns the name of the contestant with the highest score. For example, if the contestants are as shown above, a call to `findWinnerName` should return `"Harry"`.

When writing `findWinnerName`, you should assume that the contestants have not yet been sorted by score, and that there is only one contestant with the highest score. In

writing your solution, you *must* use method `sortAllRows`. You may assume that `sortAllRows` works as specified, regardless of what you wrote in part (a).

Complete method `findWinnerName`.

```
/** Returns name of contestant with highest score.
 *  Precondition:
 *   - Contestants have not been sorted by score.
 *   - Top score is unique.
 *   - Only one contestant has the highest score.
 */
public String findWinnerName()
```

STOP

END OF EXAM

ANSWER KEY
Practice Test 1

Section I

1. **C**
2. **C**
3. **D**
4. **C**
5. **E**
6. **E**
7. **A**
8. **D**
9. **B**
10. **E**
11. **C**
12. **A**
13. **B**
14. **C**
15. **B**
16. **C**
17. **D**

18. **E**
19. **E**
20. **A**
21. **B**
22. **A**
23. **D**
24. **C**
25. **B**
26. **E**
27. **C**
28. **D**
29. **B**
30. **A**
31. **C**
32. **A**
33. **E**
34. **C**
35. **E**
36. **A**
37. **D**
38. **C**
39. **A**
40. **E**

Answer Explanations

Section I

1. **(C)** Testing a program thoroughly does not prove that a program is correct. For a large program, it is generally impossible to test every possible set of input data.

2. **(C)** The private instance variable `hourlyWage` must be incremented by `amt`. Eliminate choice E, which doesn't *increment* `hourlyWage`; it simply *replaces* it by `amt`. Choice D is wrong because you can't use a method call as the left-hand side of an assignment. Choices A and B are wrong because the `incrementWage` method is void and should not return a value.

3. **(D)** The value of the boolean instance variable `isUnionMember` must be changed to the opposite of what it currently is. Segments I and II both achieve this. Note that `!true` has a value of `false` and `!false` a value of `true`. Segment III fails to do what's required if the current value of `isUnionMember` is `false`.

4. **(C)** `computePay` is a client method and so cannot access the private variables of the class. This eliminates choices A and D. The method `getHourlyWage()` must be accessed with the dot member construct; thus, choice B is wrong, and choice C is correct. Choice E is way off base—`hours` is not part of the `Worker` class, so `w.hours` is meaningless.

5. **(E)** If `s.length() < 4` for all strings in `wordList`, then `SHORT WORD` will be printed on each pass through the `for` loop. Since there are `wordList.size()` passes through the loop, the maximum number of times that `SHORT WORD` can be printed is `wordList.size()`.

6. **(E)**

$$\begin{aligned} \text{mystery}(4) &= 3 * \text{mystery}(3) \\ &= 3 * 3 * \text{mystery}(2) \\ &= 3 * 3 * 3 * \text{mystery}(1) \\ &= 3 * 3 * 3 * 3 \\ &= 81 \end{aligned}$$

7. **(A)** The declaration of the `colors` array makes the following assignments: `colors[0] = "red"`, `colors[1] = "green"`, and `colors[2] = "black"`. The loop in segment I adds these values to `colorList` in the correct order. Segment II fails because `colors` is an array and therefore can't use the `get` method. The code also confuses the lists. Segment III, in its first pass through the loop, attempts to add `colors[2]` to index position `2` of `colorList`. This will cause an `IndexOutOfBoundsException` to be thrown, since index positions `0` and `1` do not yet exist!

8. **(D)** Merge sort repeatedly splits an array of *n* elements in half until there are *n* arrays containing one element each. Now adjacent arrays are successively merged until there is a single merged, sorted array. A binary search repeatedly splits an array into two, narrowing the region that may contain the key. Insertion sort, however, does no array splitting. It takes elements one at a time and finds their insertion point in the sorted piece of the array. Elements are shifted to allow correct insertion of each element. Even though this algorithm maintains the array in two parts—a sorted part and yet-to-be-sorted part—this is not a divide-and-conquer approach.

9. **(B)** A static variable is shared by all instances of the class. "Static" means that there will be just one memory slot allocated, no matter how many `Insect`s are constructed. All instances of `Insect` access the same information stored in that slot. When an `Insect` is created, it will get tagged with the current value of `nextAvailableID` for that memory slot, which will then be incremented for the next `Insect` created. All of the other variables—`age`, `idNum`, `position`, `direction`—are specific to one instance of `Insect` and should therefore be private instance variables in the class.

10. **(E)** A new `Address` object must be created, to be used as the `Address` parameter in the `Customer` constructor. To do this correctly requires the keyword `new` preceding the `Address` constructor. Segment II omits `new` and does not use the `Address`

constructor correctly. (In fact, it inserts a new `String` object in the `Address` slot of the `Customer` constructor.)

11. **(C)** The algorithm used in method `locate` is a sequential search, which may have to examine all the objects to find the matching one. A binary search, which repeatedly discards a chunk of the array that does not contain the key, is more efficient. However, it can only be used if the values being examined—in this case customer ID numbers—are sorted. Note that it doesn't help to have the array sorted by name or phone number since the algorithm doesn't look at these values.

12. **(A)** The values of `k` are consecutively `4, 3, 2,` and `1`. The values of `randIndex` are consecutively `3, 2, 0,` and `1`. Thus, the sequence of swaps and corresponding states of `arr` will be:

```
swap arr[4] and arr[3]      1 2 3 5 4
swap arr[3] and arr[2]      1 2 5 3 4
swap arr[2] and arr[0]      5 2 1 3 4
swap arr[1] and arr[1]      5 2 1 3 4
```

13. **(B)** The `remove` method of `ArrayList` removes the indicated element, shifts the remaining elements one slot to the left (i.e., it does not leave gaps in the list), and adjusts the size of the list. Consider the list in choice B. The index values are shown:

```
The cat cat sat on the mat mat
 0   1   2   3  4   5   6   7
```

After the first occurrence of `cat` has been removed:

```
The cat sat on the mat mat
 0   1   2  3  4   5   6
```

The value of `i`, which was `1` when `cat` was removed, has now been incremented to `2` in the `for` loop. This means that the word

to be considered next is `sat`. The second occurrence of `cat` has been missed. Thus, the given code will fail whenever occurrences of the word to be removed are consecutive. You fix it by not allowing the index to increment when a removal occurs:

```
int i = 0;
while (i < wordList.size())
{
    if ((wordList.get(i)).equals(word))
        wordList.remove(i);
    else
        i++;
}
```

14. **(C)** You should not use an enhanced `for` loop to replace elements, only to access (as in segment I) or modify using a mutator method (as in segment II). Note that segment III will compile and execute, but won't replace the clocks in `allClocks` as intended.

15. **(B)** When `r` is `0`, `c` goes from `0` to `0`, and just one element, `mat[0][0]`, will be printed. When `r` is `1`, `c` goes from `0` to `1`, and two elements, `mat[1][0]` and `mat[1][1]`, will be printed, and so on. When `r` is `3`, all four elements of row `3` will be printed.

16. **(C)** To return the number of elements in the set for Method One requires no more than returning the number of elements in the array. For Method Two, however, the number of cells that contain true must be counted, which requires a test for each of the MAX values. Note that searching for a target value in the set is more efficient for Method Two. For example, to test whether 2 is in the set, simply check if `a[2] == true`. In Method One, a sequential search must be done, which is less efficient. To print all the elements in Method One, simply loop over the known number of elements and print. Method Two is less efficient because the whole array must be examined: Each cell must be tested for true before printing.

17. **(D)** An `ArithmeticException` will be thrown at run time. Note that if *N* were of type `double`, no exception would be thrown. The variable `sum` would be assigned the value `Infinity`, and the error would only be detected in the output.

18. **(E)** Only statements II and III are true. Note that *n*, the number of elements in the array, is a power of 2: $n = 64 = 2^6$. A worst case takes (exponent + 1) = 7 iterations of the search loop. Statement I is false, since the key is a left endpoint of the array, which does not represent a worst case. The key will be found in 6 iterations. Try it! Statement II, however, is true. It represents a worst case situation, in which the key is not in `arr` and is also outside the range of values of the array. So, there will be 7 passes through the loop. Statement III is true because the key, 100, is an element of `arr` that is not an endpoint. It therefore does not represent a worst case, and the key will be found in fewer than 7 iterations.

19. **(E)** The first value of `pos` is `2`, the index of the first occurrence of `"car"` in `"a carrot and car"`. Then `s` gets assigned `"carrot and car"` and `pos` is now `0`. Since `pos` is not advanced, it is stuck with a value of `0` and the method has an infinite loop. Notice that you can fix this problem by changing `s=s.substring(pos);` to `s=s.substring(pos+1);`.

20. **(A)** The correct diagram uses two up arrows to show that a `Car` *is-a* `Vehicle` and a `Truck` *is-a* `Vehicle` (inheritance relationship). The two down arrows indicate that a `Car` *has-a* `AirBag` and a `Truck` *has-a* `AirBag` (composition relationship). In each of the incorrect choices, at least one of the relationships does not make sense. For example, in choice B a `Vehicle` *has-a* `Truck`, and in choice E an `AirBag` *is-a* `Car`.

21. **(B)** The postcondition should be a true assertion about the major action of the segment. The segment overwrites the elements of array `a` with the nonnegative elements of `a`. Then `n` is adjusted so that now the array `a[0]…a[n-1]` contains just nonnegative integers. Note that even though choice E is a

correct assertion about the program segment, it's not a good postcondition because it doesn't describe the main modification to array `a` (namely, all negative integers have been removed).

22. **(A)** Note the order of precedence for the expressions involved: (1) parentheses, (2) `!`, (3) `<`, (4) `==`, (5) `&&`, (6) `||`. This means that `a < c`, `a < b`, and `!(a == c)` will all be evaluated before `||` and `&&` are considered. The given expression then boils down to `value1 || (value2 && value3)`, since `&&` has higher precedence than `||`. Notice that if `value1` is true, the whole expression is true since `(true || any)` evaluates to true. Thus, `a < c` will guarantee that the expression evaluates to true. None of the other conditions will guarantee an outcome of true. For example, suppose `a < b` (choice B). If `a == c`, then the whole expression will be false because you get `F || F`.

23. **(D)** Test data should always include a value from each range in addition to all boundary values. The given program should also handle the cases in which weights over 20 pounds or any negative weights are entered. Note that choice E contains redundant data. There is no new information to be gained in testing two weights from the same range—both 3 and 4 pounds, for example.

24. **(C)** The `max` methods shown are overloaded methods (same name but different parameter types). In the given statement, `matrix[2]` and `matrix[3]` refer to row `2` and row `3` of the matrix, respectively, each of which is an array of `int`. `max(matrix[2])` is the largest element in row `2`, namely `7`, and `max(matrix[3])` is the largest element in row `3`, namely `4`. The given statement is therefore equivalent to `max(7,4)`, which will return `7`.

25. **(B)** Segment II correctly checks that the part descriptions match and keeps track of the current part with minimum price. If this is not done, the part whose number must be returned will be lost. Segment I is incorrect because it doesn't check that `partDescription` matches the description of the current part

being examined in the array. Thus, it simply finds the `AutoPart` with the lowest price, which is not what was required. Segment III incorrectly returns the part number of the first part it finds with a matching description.

26. **(E)** Statement I is fine: The parameters are `String` objects and can be compared. Statement II shows incorrect usage of the `toString()` method, which returns strings representing the `p1` and `p2` objects. Using the dot operator and `getDescription()` following those strings is meaningless in this context, since `getDescription()` applies to the `AutoPart` class, not the `String` class. Statement III will fail because `p1` and `p2` are not `String` objects and `min` applies to strings. Also, the `AutoPart` class as currently written does not have a `compareTo` method, so `AutoPart` objects cannot be compared.

27. **(C)** Ordering of strings involves a character-by-character comparison starting with the leftmost character of each string. Thus, `strA` precedes `strB` (since `"A"` precedes `"a"`) or `strA.compareTo(strB) < 0`. This eliminates choices B and D. Eliminate choices A and E since `strB` precedes `strC` (because `"C"` precedes `"c"`) and therefore `strB.compareTo(strC) < 0`. Note that `string1.compareTo(string2) == 0` if and only if `string1` and `string2` are equal strings.

28. **(D)** Suppose `fullName` is `Dr. John Roufaiel`. In segment I, the expression `fullName.indexOf(BLANK)` returns `3`. Then, `temp` gets assigned the value of `fullName.substring(4)`, which is `John Roufaiel`. Next, `k` gets assigned the value `temp.indexOf(BLANK)`, namely `4`, and `firstName` gets assigned `temp.substring(0, 4)`, which is all the characters from `0` to `3` inclusive, namely `John`. Note that segment II works the same way, except `firstName` gets assigned `John Roufaiel` and then reassigned `John`. This is not good style, since a variable name should document its contents as precisely as possible. Still, the code works. Segment III fails because `indexOf` returns the *first* occurrence

of its `String` parameter. Thus, `firstBlank` and `secondBlank` will both contain the same value, `3`.

29. **(B)** `ThreeDigitCode` is a subclass of `ThreeDigitInteger` and therefore inherits all the public methods of `ThreeDigitInteger` except constructors. All of the statements other than B are false. For choice A, `ThreeDigitInteger` is the superclass and therefore cannot inherit from its subclass. For choice C, constructors are never inherited (see p. 143). For choice D, a subclass can access private variables of the superclass through accessor methods only (see p. 142). For choice E, a superclass cannot access any additional instance variables of its subclass.

30. **(A)** Implementation I works because it correctly calls the superclass constructor by using `super`, with the subclass parameter `aValue`, in its first line. Additionally, it correctly initializes the subclass's `isValid` private instance variable by calling the `isValid` method, which it inherits from the superclass. Implementation II is wrong because the constructor has no boolean validity parameter. Implementation III is wrong because a subclass cannot access a private instance variable of its superclass.

31. **(C)** Test III works because `newCode` is of type `ThreeDigitCode`, which has an `isValid` method. A compile-time error will occur for both tests I and II because at compile time the types of `code` and `num` are both `ThreeDigitInteger`, and the class `ThreeDigitInteger` does not have an `isValid` method. To avoid this error, the `code` object must be cast to `ThreeDigitCode`, its actual type.

32. **(A)** The *is-a* relationship must work from right-to-left: a `Parrot` *is-a* `Bird`, a `Parakeet` *is-a* `Bird`, and an `Owl` *is-a* `Bird`. All are correct. This relationship fails in declarations II and III: a `Parrot` is not necessarily a `Parakeet`, a `Bird` is not necessarily an `Owl`, and a `Bird` is not necessarily a `Parakeet`.

33. **(E)** All three segments traverse the array, accessing one element at a time, and appending it to the end of the `ArrayList`. In segment II, the first parameter of the `add` method is the position in `list` where the next string `s` will be added. Since `list.size()` increases by one after each insertion, this index is correctly updated in each pass through the enhanced `for` loop.

34. **(C)** Suppose you want random integers from 2 to 8, that is, `low` = 2 and `high` = 8. This is 7 possible integers, so you need

```
(int) (Math.random() * 7)
```

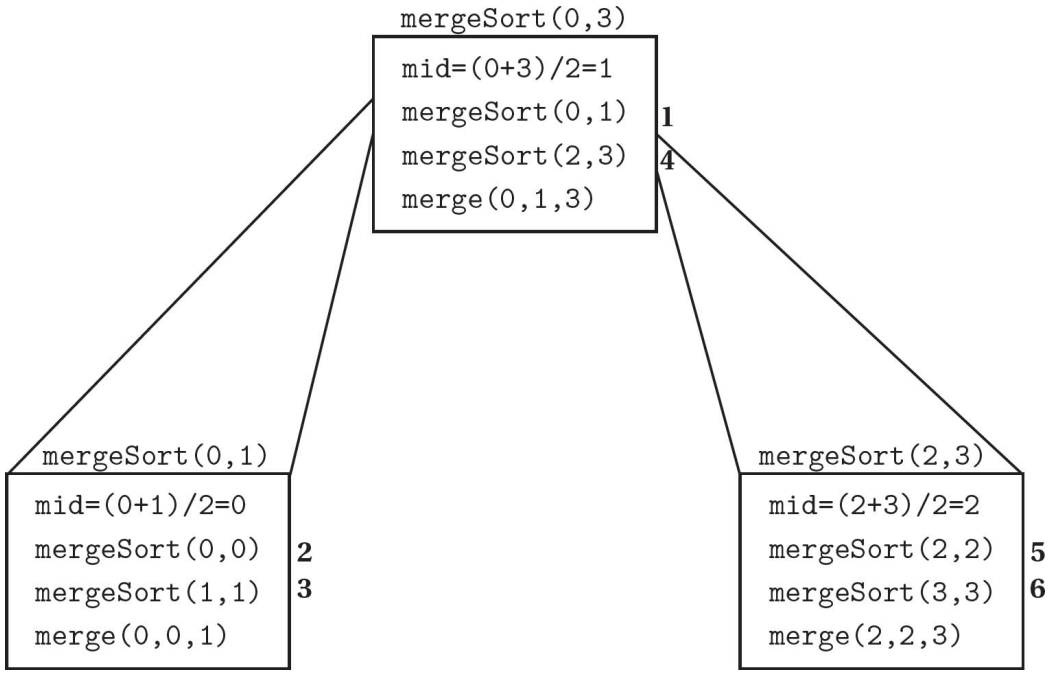
which produces 0, 1, 2, …, or 6. Therefore the quantity

```
(int) (Math.random() * 7) + 2
```

produces 2, 3, 4, …, or 8. The only expression that yields the right answer with these values is

```
(int) (Math.random() * (high - low + 1)) + low;
```

35. **(E)** Here is a "box diagram" for `mergeSort(0,3)`. The boldface numbers 1–6 show the order in which the `mergeSort` calls are made.



The `mergeSort` calls in which `first == last` are base case calls, which means that there will be no further method calls.

36. **(A)** Since the records are not sorted, the quickest way to find a given name is to start at the beginning of the list and sequentially search for that name. Choices C, D, and E will all work, but it's inefficient to sort and then search because all sorting algorithms take longer than simply inspecting each element. Choice B won't work: A binary search can only be used for a sorted list.

37. **(D)** Statement I is false because `BlockLetter` is not a subclass of `Banner`. Note that a `BlockLetter` is not a `Banner`. In the UML diagram, the down arrow indicates that a `Banner` *has-a* `BlockLetter`. Statement II is true: The down arrow of the UML diagram shows that one or more `BlockLetter` objects is used in the `Banner` code. Statement III is true: Each of the subclasses overrides the `draw` method of `BlockLetter`, the superclass. Without this feature, the program won't work as intended.

38. **(C)** The `draw` method is polymorphic, which means that it is a superclass (in this case `BlockLetter`) method that is overridden in at least one of its subclasses. During run time, there is dynamic binding between the calling object and the method; that is, the actual instance is bound to its particular overridden method. In the `drawLetters` method, the correct version of `draw` is called during each iteration of the `for` loop, and a banner with the appropriate letters is drawn.

39. **(A)** `method1` creates a mirror image of its parameter `mat` across a horizontal line placed under `mat`. If `mat` is the matrix

```
1 2 3
4 5 6
```

then the mirror image created below is

```
4 5 6
1 2 3
```

`method2` also creates a mirror image, this time with the mirror placed *above* its parameter `mat`. Note that the reflection across a horizontal line above

```
1 2 3
4 5 6
```

is also

```
4 5 6
1 2 3
```

A good general hint to solve a problem like this is to take a very simple matrix `mat` and generate some elements of `newMat`. It won't take long to see that the two methods produce the same matrix.

40. **(E)** All three changes must be made! In order to move all the `Card` elements to the temporary `ArrayList`, the `for` loop must be executed `size` times. If you start `j` at `1`, the loop will be executed `size-1` times. The error in Line 7 is subtle. With each iteration of the loop, the size of the `cards ArrayList` is being reduced by 1, so the range of random indexes is getting smaller and smaller. This won't happen if you use `size`, the length of the *original* `cards` list. You must use `cards.size()`, which is the length of the current, shorter list. If you don't make correction III, the random element will not be removed from `cards`. It will (incorrectly) remain there while a copy of it will be added to `temp`. If this error isn't corrected, execution of the method is likely to cause the `temp` list to hold more than one copy of a given card!

Section II

1. (a)
```
public static int numSymbols(String s)
{
    return s.length() - (numLetters(s) + numDigits(s));
}
```
(b)
```
public static String passwordStrength(String p)
{
    if (p.length() < 5)
        return "weak";
    else if (p.length() >= 5 && p.length() < 8)
    {
        if (numSymbols(p) > 0)
            return "medium";
        else
            return "weak";
    }
    else if (numSymbols(p) > 0 && numLetters(p) > 0 && numDigits(p) > 0)
        return "strong";
    else return "medium";
}
```

Note

- For part(b), by the first `else if` test, all passwords with fewer than 5 characters have been taken care of. By the second `else if` test, all passwords with fewer than 8 characters have been taken care of. Therefore, the second `else if` test deals only with passwords that have 8 or more characters, and you don't explicitly need to test for the number of characters.

Scoring Rubric: Password Checker

Part (a)	`numSymbols`	**2 points**
+1	use `s.length()`	
+1	subtract `(numDigits + numLetters)`	
Part (b)	`passwordStrength`	**7 points**
+1	test length < 5	
+1	test length between 5 and 8	
+1	test `numSymbols > 0`	
+1	else return `"weak"`	
+1	test for digit, letter, and symbol in long password	
+1	return `"strong"`	
+1	return `"medium"`	

2.
```
public class SnowyOwl extends Owl
{
    private boolean isMale;

    public SnowyOwl(boolean isAMale)
    {
        super("Snowy Owl");
        isMale = isAMale;
    }

    public String getColor()
    {
        if (isMale)
            return "white";
        else
            return "speckled";
    }

    public String getFood()
    {

        int num = (int) (Math.random() * 3);
        if (num == 0)
            return "hare";
        else if (num == 1)
            return "lemming";
        else
            return "small bird";
    }
}
```

Note

- The `Owl getFood` method is overridden to show the specific eating habits of a `SnowyOwl`.
- In the constructor, `super` must be used because there is no direct access to the private instance variables of the `Bird` class. The new variable `isMale` must be initialized in the constructor.

- Note that the noise for `Owl` will always be `"hoot"`. Thus, noise does not need to be provided as a parameter in the `SnowyOwl` constructor.

Scoring Rubric: Snowy Owl

	`SnowyOwl` class	**9 points**
+1	use of `extends` in declaration	
+1	constructor declaration	
+1	code using `super`	
+1	`"Snowy owl"` parameter	
+1	declaration of `getFood`	
+1	`getColor` method	
+1	get random `int` from 0 to 2	
+1	`if…else` statement	
+1	`return` statements	

3. (a)

```
public String extractCity(String cityZip)
{
    int commaPos = cityZip.indexOf(",");
    return cityZip.substring(0, commaPos);
}
```

(b)

```
public String getAddress(String name)
{
    int index = 0;
    while(index < lines.size() && !name.equals(lines.get(index)))
        index++;
    index++;
    String s = "";
    while (!(lines.get(index).equals("")))
    {
        s += lines.get(index) + "\n";
        index++;
    }
    return s;
}
```

Note

- Part (b) first finds the name that matches the parameter, and then builds a string out of the next two or three lines that comprise the address. Again, the empty string signals that the end of the address has been reached.
- The escape character string, `"\n"`, inserts a line break into the string.

Scoring Rubric: Mailing List

Part (a)	`extractCity`	3 **points**
+1	locate comma in ZIP code	
+1	return substring that contains the city	
+1	substring parameters	
Part (b)	`getAddress`	6 **points**
+1	loop to search for name	
+1	loop to traverse `lines` until end of address is reached	
+1	check range of `index`	
+1	concatenate lines of address	
+1	newline characters	
+1	return string containing address	

4. (a)

```
public void sortAllRows()
{
    for(Contestant[] row: contestants)
        sort(row);
}
```

(b)
```
public String findWinnerName()
{
    sortAllRows();
    int max = contestants[0][0].getScore();
    String winner = contestants[0][0].getName();
    for(int k = 0; k < NUM_ROWS; k++)
    {
        Contestant c = contestants[k][CONTESTANTS_PER_ROW - 1];
        if (c.getScore() > max)
        {
            winner = c.getName();
            max = c.getScore();
        }
    }
    return winner;
}
```

Note

- Part (a) uses the Java feature that a two-dimensional array is an array of arrays. Thus, each row, which is an array of `Contestant`, can be sorted using the helper method `sort`.
- Part (b) uses the fact that after you sort all the rows of contestants, the winning contestant will be in the last column of the matrix of contestants. When you go through the loop, searching for a score that's higher than the current max, be sure to store the name that goes with that score!

Scoring Rubric: Two-Dimensional Contest Organizer

Part (a)	`sortAllRows`	**2 points**
+1	loop over the rows of `Contestants`	
+1	sort each row	
Part (b)	`findWinnerName`	**7 points**

+1	call `sortAllRows`
+1	get score of top scorer in first row
+1	get name of top scorer in first row
+1	loop over all rows
+1	accessing contestant in last column
+1	test for score higher than `max`
+1	adjust winner and max if higher score found

How to Calculate Your (Approximate) AP Computer Science A Score

Multiple-Choice

Number correct (out of 40) = _____ ⟸ Multiple-Choice Score

Free-Response

Question 1 _____ (out of 9)

Question 2 _____ (out of 9)

Question 3 _____ (out of 9)

Question 4 _____ (out of 9)

Total _____ × 1.11 = _____ ⟸ Free-Response Score (Do not round.)

Final Score

Multiple-Choice Score	+	Free-Response Score	=	Final Score (Round to nearest whole number.)
_____		_____		_____

Chart to Convert to AP Grade Computer Science A

Final Score Range	AP Grade[a]
62–80	5

47–61	4
37–46	3
29–36	2
0–28	1

[a]The score range corresponding to each grade varies from exam to exam and is approximate.