Week 22

AP Computer Science A

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

4. Data Collections

Starter Quiz · 课前小测

AP Computer Science A Name: _____ Score: _____

1. A program that runs correctly is...

   ☐ not automatically ethical

   ☐ always ethical

   ☐ always private

   ☐ never biased

2. A structure that stores many values of the same type under one name is a(n)...

   ☐ array

   ☐ int

   ☐ boolean

   ☐ constructor

3. For int[] a = {10, 20, 30}; what is a[0]?

   ☐ 10

   ☐ 20

   ☐ 30

   ☐ there is no a[0]

4. The correct condition to traverse an array a with an index is...

   ☐ i < a.length

   ☐ i <= a.length

   ☐ i < a.length()

   ☐ i > 0

5. To find the maximum of an array, you should initialize max to...

   ☐ the first element a[0]

   ☐ 0

   ☐ the array length

   ☐ -1

6. For a = {3, 9, 5}, what is the maximum value?

   _____

7. A forward for loop with i++ and remove(i) skips the element after each removal.

   ☐ True ☐ False

8. An algorithm trained on biased data can amplify that unfairness at scale.

   ☐ True ☐ False

9. For int[] a = {10, 20, 30}; what is a.length?

   _____

10. In for (int x : a) { x = 0; }, the array a is set to all zeros.

    ☐ True ☐ False

AP Computer Science A

1. **not automatically ethical**

   Correct code can still violate privacy or amplify bias.

2. **array**

   An array is a fixed-size, same-type collection.

3. **10**

   Arrays are indexed from 0, so a[0] is the first element, 10.

4. **i < a.length**

   Valid indices are 0..length-1, so i < a.length.

5. **the first element a[0]**

   Starting at 0 fails if all values are negative.

6. **9**

   9 is the largest element.

7. **True**

   The shifted neighbor takes index i but is never checked.

8. **True**

   Bias in data can produce unjust outcomes for many people.

9. **3**

   Three elements → length 3.

10. **False**

    x is a local copy; the array is unchanged.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.1 Ethical and Social Issues Around Data Collection

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS data 数据 • privacy 隐私 • bias 偏见 • consent 同意 • array 数组

Objective

Build the skills to answer exam questions on **ethical and social issues around data collection**.

You must be able to:

- describe how programs collect, store, and use large amounts of **data** 数据
- explain why **privacy** 隐私 matters when personal data is gathered
- discuss the risk of **bias** 偏见 when a data set is unrepresentative

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Data and privacy

Programs can collect, store, and process **huge** amounts of data. When that data is **personal**, **privacy** matters — it can be misused if exposed or shared without consent.

■ Bias in data

If a data set does **not represent everyone fairly**, conclusions drawn from it can be **biased**, harming under-represented groups.

2 Practice

2.1 State why privacy matters when collecting personal data. [1]

2.2 State one risk if a data set does not represent everyone fairly. [1]

2.3 State one responsibility when handling collected data. [1]

3 Exam-style questions

3.1 A data set that leaves out some groups can introduce [1]

- **A** speed
- **B** bias
- **C** encryption
- **D** an array

3.2 Collecting personal data raises concerns about [1]

- **A** privacy
- **B** compilation
- **C** casting
- **D** recursion

3.3 An app collects users' location data.

(a) State one privacy concern. [1]

(b) State one risk of an unrepresentative data set. [1]

(c) State one safe practice for handling the data. [1]

Solutions

2.1 personal data can be misused if it is exposed or shared without consent.

2.2 the conclusions can be biased and unfair to under-represented groups.

2.3 protect privacy, store data securely, or use representative data (any one).

3.1 B.

3.2 A.

3.3 (a) users' movements could be tracked or exposed. (b) biased or unfair results. (c) collect only what is needed, secure it, and get consent (any one).

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.2 Introduction to Using Data Sets

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS collection 集合 • data set 数据集 • array 数组

Objective

Build the skills to answer exam questions on **using data sets**.

You must be able to:

- understand that a **data set** 数据集 is a collection of related values processed together
- explain why a **collection** 集合 is more convenient than many separate variables
- describe common operations: storing, reading, and summarizing values

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Data sets and collections

A **data set** is a collection of related values handled together. A **collection** (such as an array) is far more convenient than many separate variables — one loop can process all the values.

■ Common operations

Store values, **read** them back, and **summarize** them (sum, average, maximum, count).

2 Practice

2.1 Define a data set. [1]

2.2 State why a collection is more convenient than many separate variables. [1]

2.3 Name one common operation on a data set. [1]

3 Exam-style questions

3.1 A data set is a collection of [1]

- **A** unrelated values
- **B** related values processed together
- **C** methods
- **D** classes

3.2 Storing 100 scores is easier with [1]

- **A** 100 separate variables
- **B** one collection
- **C** no storage at all
- **D** a constructor

3.3 A program stores the daily temperatures for a month.

(a) Name a structure that could hold them. [1]

(b) State one operation you might perform on them. [1]

(c) State one benefit over using separate variables. [1]

Solutions

2.1 a collection of related values processed together.

2.2 one loop can process every value, instead of naming each separately.

2.3 storing, reading, or summarizing values (any one).

3.1 B.

3.2 B.

3.3 (a) an array (or list). (b) find the sum, average, or maximum (any one). (c) all values can be processed with a single loop.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.3 Array Creation and Access

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS array 数组 • index 索引

Objective

Build the skills to answer exam questions on **array creation and access**.

You must be able to:

- declare and create an **array** 数组 to store a fixed number of same-type values
- use an **index** 索引 to access an element, starting from 0
- find the number of elements with the `length` field

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Creating and accessing an array

```
int[] a = new int[5];   // 5 ints, indices 0..4
int[] nums = {10, 20, 30, 40};
```

Access an element by **index** (from 0): `nums[0]` is `10`, and the last is `nums[nums.length - 1]`.

■ Length

`nums.length` gives the number of elements (here 4). Note it is a **field**, not a method — no parentheses.

2 Practice

2.1 State the first index of an array. [1]

2.2 For `int[] a = new int[6];`, state `a.length`. [1]

2.3 State how to access the third element of array `a`. [1]

3 Exam-style questions

3.1 Array indices start at [1]

- **A** `1`
- **B** `0`
- **C** `-1`
- **D** `length`

3.2 The number of elements in array `a` is [1]

- **A** `a.size()`
- **B** `a.length`
- **C** `a.count`
- **D** `length(a)`

3.3 `int[] nums = {10, 20, 30, 40};`.

(a) State `nums[0]`. [1]

(b) State `nums.length`. [1]

(c) State the last valid index. [1]

Solutions

2.1 `0`.

2.2 `6`.

2.3 `a[2]` (index 2 is the third element).

3.1 B.

3.2 B.

3.3 (a) `10`. (b) `4`. (c) `3`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.4 Array Traversals

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS enhanced for loop 增强 • traverse 遍历 • array 数组

Objective

Build the skills to answer exam questions on **array traversals**.

You must be able to:

- **traverse** 遍历 an array with a `for` loop that visits every index
- use an **enhanced for loop** 增强 for 循环 (for-each) to read each element
- understand that a for-each loop **cannot replace** elements in the array

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Standard for traversal

```
for (int i = 0; i < a.length; i++) {
    System.out.println(a[i]);   // has the index i
}
```

■ Enhanced for (for-each)

```
for (int x : a) {
    System.out.println(x);   // reads each element
}
```

A for-each loop is simpler for **reading**, but it **cannot replace** the elements in the array — for that you need the index.

2 Practice

2.1 State the loop range used to visit every index of an array `a`. [1]

2.2 State what an enhanced for loop **cannot** do. [1]

2.3 State how many elements a for-each loop over array `a` visits. [1]

---

3 Exam-style questions

3.1 An enhanced for loop (for-each) is used to [1]

- **A** replace elements
- **B** read each element
- **C** create the array
- **D** sort the array

3.2 A standard for loop over array `a` runs from `0` to [1]

- **A** `a.length`
- **B** `a.length - 1`
- **C** `1`
- **D** `a.length + 1`

3.3 `int[] a = {3, 6, 9};`.

(a) Write the loop condition for a standard for loop over `a` (using `i`). [1]

(b) State whether a for-each loop can change `a`'s values. [1]

(c) State how many elements a for-each loop visits. [1]

Solutions

2.1 from `0` to `a.length - 1`.

2.2 replace (change) the elements of the array.

2.3 all of them (here 3).

3.1 B.

3.2 B.

3.3 (a) `i < a.length`. (b) no. (c) `3`.

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (`"."`), exclamation point (`"!"`), or letter, or is a `String` of length `0` if the user did not enter a comment.

```
public class Review
{
   private int rating;
   private String comment;

   /** Precondition: r >= 0
    *      c is not null.
    */
   public Review(int r, String c)
   {
      rating = r;
      comment = c;
   }

   public int getRating()
   {
      return rating;
   }

   public String getComment()
   {
      return comment;
   }

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

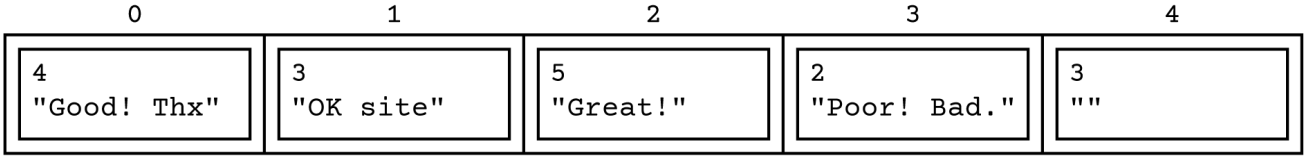
GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    {  /* implementation not shown */  }

    /** Returns a double representing the average rating of all the Review objects to be
     *  analyzed, as described in part (a)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     */
    public double getAverageRating()
    {  /* to be implemented in part (a) */  }

    /** Returns an ArrayList of String objects containing formatted versions of
     *  selected user comments, as described in part (b)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     *  Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return `3.4` if `allReviews` contained the following `Review` objects.



Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 *  analyzed, as described in part (a)
 *  Precondition: allReviews contains at least one Review.
 *      No element of allReviews is null.
 */
public double getAverageRating()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

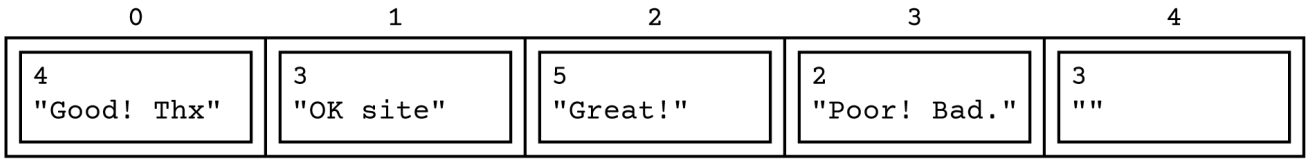
```
public class Review

private int rating
private String comment

public Review(int r, String c)
public int getRating()
public String getComment()

public class ReviewAnalysis

private Review[] allReviews

public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

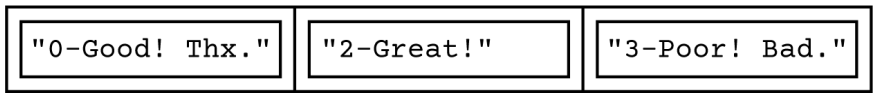
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (`"-"`).
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).



The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index `1` and index `4` in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.



Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 *  selected user comments, as described in part (b)
 *  Precondition: allReviews contains at least one Review.
 *       No element of allReviews is null.
 *  Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

3. A high school club maintains information about its members in a `MemberInfo` object. A `MemberInfo` object stores a club member's name, year of graduation, and whether or not the club member is in *good standing*. A member who is in good standing has fulfilled all the responsibilities of club membership.

   A partial declaration of the `MemberInfo` class is shown below.

```
public class MemberInfo
{
   /** Constructs a MemberInfo object for the club member with name name,
    *  graduation year gradYear, and standing hasGoodStanding.
    */
   public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
   {  /* implementation not shown */  }

   /** Returns the graduation year of the club member. */
   public int getGradYear()
   {  /* implementation not shown */  }

   /** Returns true if the member is in good standing and false otherwise. */
   public boolean inGoodStanding()
   {  /* implementation not shown */  }

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

   The `ClubMembers` class maintains a list of current club members. The declaration of the `ClubMembers` class is shown below.

```
public class ClubMembers
{
   private ArrayList<MemberInfo> memberList;

   /** Adds new club members to memberList, as described in part (a).
    *  Precondition: names is a non-empty array.
    */
   public void addMembers(String[] names, int gradYear)
   {  /* to be implemented in part (a) */  }

   /** Removes members who have graduated and returns a list of members who have graduated
    *  and are in good standing, as described in part (b).
    */
   public ArrayList<MemberInfo> removeMembers(int year)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `ClubMembers` method `addMembers`, which takes two parameters. The first parameter is a `String` array containing the names of new club members to be added. The second parameter is the graduation year of all the new club members. The method adds the new members to the `memberList` instance variable. The names can be added in any order. All members added are initially in good standing and share the same graduation year, `gradYear`.

Complete the `addMembers` method.

```
/** Adds new club members to memberList, as described in part (a).
 *  Precondition: names is a non-empty array.
 */
public void addMembers(String[] names, int gradYear)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `ClubMembers` method `removeMembers`, which takes the following actions.

- Returns a list of all students who have graduated and are in good standing. A member has graduated if the member's graduation year is less than or equal to the method's `year` parameter. If no members meet these criteria, an empty list is returned.
- Removes from `memberList` all members who have graduated, regardless of whether or not they are in good standing.

The following example illustrates the results of a call to `removeMembers`.

The `ArrayList memberList` before the method call `removeMembers(2018)`:

"SMITH, JANE"	"FOX, STEVE"	"XIN, MICHAEL"	"GARCIA, MARIA"
2019	2018	2017	2020
false	true	false	true

The `ArrayList memberList` after the method call `removeMembers(2018)`:

"SMITH, JANE"	"GARCIA, MARIA"
2019	2020
false	true

The `ArrayList` returned by the method call `removeMembers(2018)`:

"FOX, STEVE"
2018
true

Complete the `removeMembers` method.

```
/** Removes members who have graduated and returns a list of members who have graduated and are
 *  in good standing, as described in part (b).
 */
public ArrayList<MemberInfo> removeMembers(int year)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class MemberInfo

public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
public int getGradYear()
public boolean inGoodStanding()

public class ClubMembers

private ArrayList<MemberInfo> memberList

public void addMembers(String[] names, int gradYear)
public ArrayList<MemberInfo> removeMembers(int year)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the Java 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 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`.

	`arr1`				Value returned by `arraySum(arr1)`
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (c) begins on page 6.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.5 Implementing Array Algorithms

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS array 数组

Objective

Build the skills to answer exam questions on **implementing array algorithms**.

You must be able to:

- find the **sum**, **average**, minimum, or maximum of an array
- **count** how many elements meet a condition
- determine whether an array **contains** a matching value

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Sum and average

```
int sum = 0;
for (int x : a) sum += x;
double avg = (double) sum / a.length;
```

■ Minimum, maximum, and counting

Track a "best so far" variable for the min or max; use an `if` inside the loop to **count** elements that meet a condition, or to detect whether a value is present.

2 Practice

2.1 Describe how to find the sum of an array. [1]

2.2 For `int[] a = {2, 4, 6};`, find the sum and the average. [2]

2.3 State how to count the elements greater than 5. [1]

3 Exam-style questions

3.1 To find the maximum of an array, you [1]

- **A** add all the elements
- **B** track the largest value seen so far
- **C** sort then divide
- **D** count the elements

3.2 The average of `{4, 8, 12}` is [1]

- **A** `8`
- **B** `24`
- **C** `4`
- **D** `12`

3.3 `int[] a = {5, 10, 15, 20};`.

(a) Find the sum. [1]

(b) Find the average. [1]

(c) Count the elements greater than 10. [1]

Solutions

2.1 start a total at 0 and add each element as you loop through the array.

2.2 sum $= 2 + 4 + 6 = 12$; average $= 12/3 = 4$.

2.3 loop over the array and increment a counter inside an `if (x > 5)`.

3.1 B.

3.2 A.

3.3 (a) 50. (b) 12.5. (c) 2 (the values 15 and 20).

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (`"."`), exclamation point (`"!"`), or letter, or is a `String` of length `0` if the user did not enter a comment.

```
public class Review
{
   private int rating;
   private String comment;

   /** Precondition: r >= 0
    *      c is not null.
    */
   public Review(int r, String c)
   {
      rating = r;
      comment = c;
   }

   public int getRating()
   {
      return rating;
   }

   public String getComment()
   {
      return comment;
   }

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

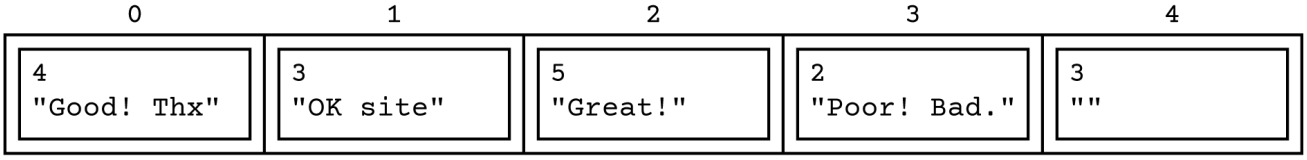
GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    {  /* implementation not shown */  }

    /** Returns a double representing the average rating of all the Review objects to be
     *  analyzed, as described in part (a)
     *  Precondition: allReviews contains at least one Review.
     *       No element of allReviews is null.
     */
    public double getAverageRating()
    {  /* to be implemented in part (a) */  }

    /** Returns an ArrayList of String objects containing formatted versions of
     *  selected user comments, as described in part (b)
     *  Precondition: allReviews contains at least one Review.
     *       No element of allReviews is null.
     *  Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return `3.4` if `allReviews` contained the following `Review` objects.



Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 *  analyzed, as described in part (a)
 *  Precondition: allReviews contains at least one Review.
 *      No element of allReviews is null.
 */
public double getAverageRating()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

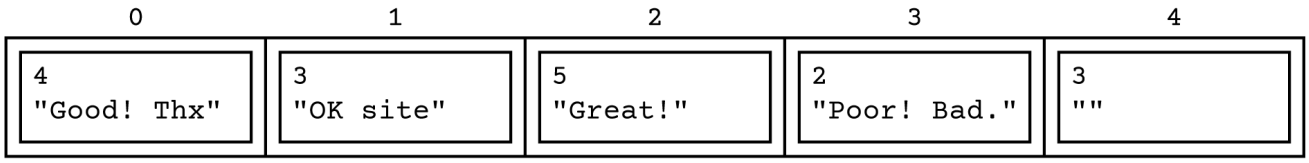
```
public class Review

private int rating
private String comment

public Review(int r, String c)
public int getRating()
public String getComment()

public class ReviewAnalysis

private Review[] allReviews

public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

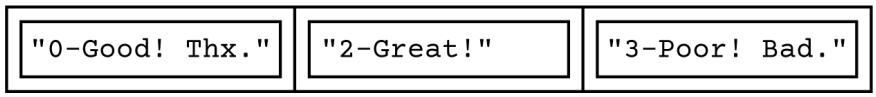
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (`"-"`).
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).



The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index `1` and index `4` in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.



Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 *  selected user comments, as described in part (b)
 *  Precondition: allReviews contains at least one Review.
 *       No element of allReviews is null.
 *  Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

3. Many encoded strings contain *delimiters*. A delimiter is a non-empty string that acts as a boundary between different parts of a larger string. The delimiters involved in this question occur in pairs that must be *balanced*, with each pair having an open delimiter and a close delimiter. There will be only one type of delimiter for each string. The following are examples of delimiters.

   Example 1
   Expressions in mathematics use open parentheses `"("` and close parentheses `")"` as delimiters. For each open parenthesis, there must be a matching close parenthesis.

`(x + y) * 5`	is a valid mathematical expression.
`(x + (y)`	is NOT a valid mathematical expression because there are more open delimiters than close delimiters.

   Example 2
   HTML uses `<B>` and `</B>` as delimiters. For each open delimiter `<B>`, there must be a matching close delimiter `</B>`.

`<B> Make this text bold </B>`	is valid HTML.
`<B> Make this text bold </UB>`	is NOT valid HTML because there is one open delimiter and no matching close delimiter.

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

    /** The open and close delimiters. */
    private String openDel;
    private String closeDel;

    /** Constructs a Delimiters object where open is the open delimiter and close is the
     *  close delimiter.
     *  Precondition: open and close are non-empty strings.
     */
    public Delimiters(String open, String close)
    {
       openDel = open;
       closeDel = close;
    }

    /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
    public ArrayList<String> getDelimitersList(String[] tokens)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
     *  Precondition: delimiters contains only valid open and close delimiters.
     */
    public boolean isBalanced(ArrayList<String> delimiters)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

`openDel:` `"("`

`closeDel:` `")"`

`tokens:`

`"("`	`"x + y"`	`")"`	`" * 5"`

`ArrayList` of delimiters:

`"("`	`")"`

Example 2

`openDel:` `"<q>"`

`closeDel:` `"</q>"`

`tokens:`

`"<q>"`	`"yy"`	`"</q>"`	`"zz"`	`"</q>"`

`ArrayList` of delimiters:

`"<q>"`	`"</q>"`	`"</q>"`

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

(b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1
The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

"^{"	"^{"	"}"	"^{"	"}"	"}"

Example 2
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"^{"	"}"	"</sup>"	"<sup>"

↑

When starting from the left, at this point, condition 1 is violated.

Example 3
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"</sup>"

↑

At this point, condition 1 is violated.

Example 4
The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.

"<sup>"	"^{"	"}"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 30 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the Java 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 involves the implementation and extension of a `RandomStringChooser` class.

   (a) A `RandomStringChooser` object is constructed from an array of non-`null` `String` values. When the object is first constructed, all of the strings are considered available. The `RandomStringChooser` class has a `getNext` method, which has the following behavior. A call to `getNext` returns a randomly chosen string from the available strings in the object. Once a particular string has been returned from a call to `getNext`, it is no longer available to be returned from subsequent calls to `getNext`. If no strings are available to be returned, `getNext` returns `"NONE"`.

   The following code segment shows an example of the behavior of `RandomStringChooser`.

```
String[] wordArray = {"wheels", "on", "the", "bus"};
RandomStringChooser sChooser = new RandomStringChooser(wordArray);
for (int k = 0; k < 6; k++)
{
   System.out.print(sChooser.getNext() + " ");
}
```

   One possible output is shown below. Because `sChooser` has only four strings, the string `"NONE"` is printed twice.

```
bus the wheels on NONE NONE
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

Part (b) begins on page 4.

(b) The following partially completed `RandomLetterChooser` class is a subclass of the `RandomStringChooser` class. You will write the constructor for the `RandomLetterChooser` class.

```
public class RandomLetterChooser extends RandomStringChooser
{
    /** Constructs a random letter chooser using the given string str.
     *  Precondition: str contains only letters.
     */
    public RandomLetterChooser(String str)
    {  /* to be implemented in part (b) */  }

    /** Returns an array of single-letter strings.
     *  Each of these strings consists of a single letter from str. Element k
     *  of the returned array contains the single letter at position k of str.
     *  For example, getSingleLetters("cat") returns the
     *  array { "c", "a", "t" }.
     */
    public static String[] getSingleLetters(String str)
    {  /* implementation not shown */  }
}
```

The following code segment shows an example of using `RandomLetterChooser`.

```
RandomLetterChooser letterChooser = new RandomLetterChooser("cat");
for (int k = 0; k < 4; k++)
{
   System.out.print(letterChooser.getNext());
}
```

The code segment will print the three letters in `"cat"` in one of the possible orders. Because there are only three letters in the original string, the code segment prints `"NONE"` the fourth time through the loop. One possible output is shown below.

```
actNONE
```

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 *  Precondition: str contains only letters.
 */
public RandomLetterChooser(String str)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the Java 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 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`.

	`arr1`				Value returned by `arraySum(arr1)`
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (c) begins on page 6.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.6 Using Text Files

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS text file 文本文件 • scanner 扫描器

Objective

Build the skills to answer exam questions on **using text files**.

You must be able to:

- use the **Scanner** 扫描器 class to read data from a **text file** 文本文件
- open a file with a `File` object and read it line by line
- use methods such as `hasNextLine`, `nextLine`, and `nextInt`

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Reading a file with a Scanner

```
Scanner in = new Scanner(new File("data.txt"));
while (in.hasNextLine()) {
    String line = in.nextLine();
}
```

■ Useful methods

- `hasNextLine()` — `true` while there is another line to read.
- `nextLine()` — reads and returns the next line.
- `nextInt()` — reads the next integer.

2 Practice

2.1 Name the class commonly used to read a text file. [1]

2.2 State what `hasNextLine()` returns. [1]

2.3 Name a method that reads one line of a file. [1]

3 Exam-style questions

3.1 To read a text file line by line, you loop while [1]

- **A** `hasNextLine()`
- **B** `nextLine()`
- **C** `length`
- **D** `size()`

3.2 `nextInt()` reads [1]

- **A** a whole line
- **B** the next integer
- **C** an entire file
- **D** a boolean

3.3 A program reads numbers from a file.

(a) Name the class used. [1]

(b) Name a method that checks whether there is more input. [1]

(c) Name a method that reads the next integer. [1]

Solutions

2.1 `Scanner`.

2.2 `true` if there is another line to read, otherwise `false`.

2.3 `nextLine()`.

3.1 **A**.

3.2 **B**.

3.3 (a) `Scanner`. (b) `hasNextLine()` (or `hasNextInt()`). (c) `nextInt()`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.7 Wrapper Classes

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS wrapper class 包装类 • unboxing 拆箱 • autoboxing 自动装箱

Objective

Build the skills to answer exam questions on **wrapper classes**.

You must be able to:

- understand that a **wrapper class** 包装类 lets a primitive be treated as an object
- use the `Integer` and `Double` classes
- describe **autoboxing** 自动装箱 and **unboxing** 拆箱

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Wrapper classes

A **wrapper class** lets a primitive value be treated as an **object**: `Integer` wraps an `int`, and `Double` wraps a `double`. This is needed because some structures (like `ArrayList`) store only objects.

■ Autoboxing and unboxing

```
Integer n = 5;   // autoboxing: int -> Integer
int m = n;       // unboxing: Integer -> int
```

2 Practice

2.1 State what a wrapper class does. [1]

2.2 Name the wrapper class for `int`. [1]

2.3 State the difference between autoboxing and unboxing. [2]

3 Exam-style questions

3.1 The wrapper class for `double` is [1]

- **A** `Double`
- **B** `double`
- **C** `Number`
- **D** `Float`

3.2 Converting an `int` to an `Integer` automatically is [1]

- **A** unboxing
- **B** autoboxing
- **C** casting
- **D** overloading

3.3 `Integer x = 7;`.

(a) Name what happened to the `7`. [1]

(b) Name the wrapper class used. [1]

(c) Name the reverse process (`Integer` to `int`). [1]

Solutions

2.1 it lets a primitive value be treated as an object.

2.2 `Integer`.

2.3 autoboxing converts a primitive to its wrapper; unboxing converts a wrapper back to a primitive.

3.1 A.

3.2 B.

3.3 (a) it was autoboxed into an `Integer`. (b) `Integer`. (c) unboxing.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.8 ArrayList Methods

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS arraylist 动态数组 • array 数组

Objective

Build the skills to answer exam questions on **ArrayList methods**.

You must be able to:

- create an **ArrayList** 动态数组 that can grow and shrink
- use `add`, `get`, `set`, `remove`, and `size`
- understand that an ArrayList stores **objects** (primitives as wrappers)

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ ArrayList

Unlike an array, an **ArrayList** can grow and shrink as elements are added or removed:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(5);        // [5]
list.get(0);        // returns 5
list.size();        // 1
```

- `add`, `get`, `set`, `remove`, and `size` are the core methods.
- An ArrayList stores **objects**, so primitives are stored as wrappers (an `int` becomes an `Integer`).

■ A worked sequence

Start empty, then:

```
list.add(5); list.add(8);   // [5, 8]
list.set(0, 3);             // [3, 8]
list.remove(1);             // [3]
```

`remove(1)` deletes the element at **index 1** (the `8`), leaving `[3]`, so `list.size()` is now `1`.

2 Practice

2.1 State one way an ArrayList differs from an array. [1]

2.2 Name the method that returns the number of elements. [1]

2.3 State what `list.get(0)` returns. [1]

3 Exam-style questions

3.1 To add an element to an ArrayList, you use [1]

- **A** `add`
- **B** `length`
- **C** `new`
- **D** `size`

3.2 The number of elements in an ArrayList is found with [1]

- **A** `.length`
- **B** `.size()`
- **C** `.count`
- **D** `.get()`

3.3 `ArrayList<String> a = new ArrayList<>(); a.add("hi"); a.add("bye");`.

(a) State `a.size()`. [1]

(b) State `a.get(1)`. [1]

(c) State whether an ArrayList can grow. [1]

Solutions

2.1 an ArrayList can grow and shrink, while an array has a fixed size.

2.2 `size()`.

2.3 the first element of the list.

3.1 A.

3.2 B.

3.3 (a) `2`. (b) `"bye"`. (c) yes.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.9 ArrayList Traversals

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS traverse 遍历 • array 数组 • arraylist 动态数组

Objective

Build the skills to answer exam questions on **ArrayList traversals**.

You must be able to:

- **traverse** 遍历 an ArrayList with an indexed `for` loop from 0 to `size - 1`
- use an **enhanced for loop** to read each element
- add or remove elements safely while looping

1 Worked examples

Study these first. Each one shows the method for a question type used later.

Two ways to traverse

```
for (int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i));
}
for (int x : list) {
    System.out.println(x);
}
```

Modifying while looping

Removing an element during an indexed loop shifts the later elements down, so it is easy to **skip** one — adjust the index carefully or loop backwards.

2 Practice

2.1 State the index range for an indexed traversal of an ArrayList. [1]

2.2 Name the method that gives an ArrayList's number of elements. [1]

2.3 State one risk of removing elements while looping forward with an index. [1]

3 Exam-style questions

3.1 An indexed for loop over an ArrayList runs from `0` to [1]

- **A** `size`
- **B** `size - 1`
- **C** `length`
- **D** `length - 1`

3.2 To read each element of an ArrayList simply, use [1]

- **A** a for-each loop
- **B** a constructor
- **C** a cast
- **D** a static method

3.3 An `ArrayList` has 4 elements.

(a) State the index of the first element. [1]

(b) State the index of the last element. [1]

(c) Name the method giving the number of elements. [1]

Solutions

2.1 from `0` to `size() - 1`.

2.2 `size()`.

2.3 the later elements shift down, so the next one can be skipped.

3.1 B.

3.2 A.

3.3 (a) `0`. (b) `3`. (c) `size()`.

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
   /** Initialized in the constructor and contains no null elements */
   private ArrayList<String> wordList;

   /**
    *  Returns true if each element of wordList (except the first) contains the previous
    *  element as a substring and returns false otherwise, as described in part (a)
    *  Precondition: wordList contains at least two elements.
    *  Postcondition: wordList is unchanged.
    */
   public boolean isWordChain()
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns an ArrayList<String> based on strings from wordList that start
    *  with target, as described in part (b). Each element of the returned ArrayList has had
    *  the initial occurrence of target removed.
    *  Postconditions: wordList is unchanged.
    *       Items appear in the returned list in the same order as they appear in wordList.
    */
   public ArrayList<String> createList(String target)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

`wordList`	`isWordChain` **Return Value**	**Explanation**
`["an", "band", "band", "abandon"]`	`true`	Each element contains the previous element as a substring.
`["to", "too", "stool", "tools"]`	`false`	`"tools"` does not contain the substring `"stool"`.

Complete the `isWordChain` method.

```
/**
 *  Returns true if each element of wordList (except the first) contains the previous
 *  element as a substring and returns false otherwise, as described in part (a)
 *  Precondition: wordList contains at least two elements.
 *  Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	`ArrayList` **Returned by** `createList`	**Explanation**
`createList("cat")`	`["ch", "chacat", ""]`	Only `"catch"`, `"catchacat"`, and `"cat"` begin with `"cat"`.
`createList("catch")`	`["", "acat"]`	Only `"catch"` and `"catchacat"` begin with `"catch"`.
`createList("dog")`	`[]`	None of the words in `wordList` begin with `"dog"`.

Complete the `createList` method.

```
/**
 *  Returns an ArrayList<String> based on strings from wordList that start
 *  with target, as described in part (b). Each element of the returned ArrayList has had
 *  the initial occurrence of target removed.
 *  Postconditions: wordList is unchanged.
 *      Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

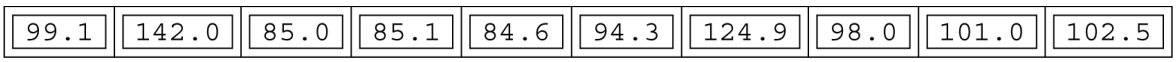
```
public class WordChecker

private ArrayList<String> wordList

public boolean isWordChain()
public ArrayList<String> createList(String target)
```

3. This question involves the analysis of weather data. The following `WeatherData` class has an instance variable, `temperatures`, which contains the daily high temperatures recorded on consecutive days at a particular location. The class also contains methods used to analyze that data. You will write two methods of the `WeatherData` class.

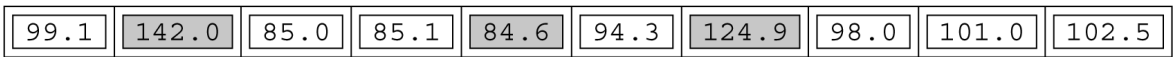
```
public class WeatherData
{
    /** Guaranteed not to be null and to contain only non-null entries */
    private ArrayList<Double> temperatures;

    /**
     *  Cleans the data by removing from temperatures all values that are less than
     *  lower and all values that are greater than upper, as described in part (a)
     */
    public void cleanData(double lower, double upper)
    {  /* to be implemented in part (a) */  }

    /**
     *  Returns the length of the longest heat wave found in temperatures, as described in
     *  part (b)
     *  Precondition: There is at least one heat wave in temperatures based on threshold.
     */
    public int longestHeatWave(double threshold)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `cleanData` method, which modifies the `temperatures` instance variable by removing all values that are less than the `lower` parameter and all values that are greater than the `upper` parameter. The order of the remaining values in `temperatures` must be maintained.

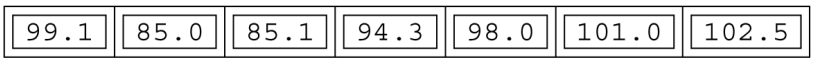
For example, consider a `WeatherData` object for which `temperatures` contains the following.



The three shaded values shown would be removed by the method call `cleanData(85.0, 120.0)`.



The following shows the contents of `temperatures` after the three shaded values are removed as a result of the method call `cleanData(85.0, 120.0)`.



Complete method `cleanData`.

```
/**
 *  Cleans the data by removing from temperatures all values that are less than
 *  lower and all values that are greater than upper, as described in part (a)
 */
public void cleanData(double lower, double upper)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `longestHeatWave` method, which returns the length of the longest heat wave found in the `temperatures` instance variable. A heat wave is a sequence of two or more consecutive days with a daily high temperature greater than the parameter `threshold`. The `temperatures` instance variable is guaranteed to contain at least one heat wave based on the `threshold` parameter.

For example, consider the following contents of `temperatures`.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

In the following sample contents of `temperatures`, all heat waves based on the `threshold` temperature of `100.5` are shaded. The method call `longestHeatWave(100.5)` would return `3`, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

In the following sample contents of `temperatures`, all heat waves based on the `threshold` temperature of `95.2` are shaded. The method call `longestHeatWave(95.2)` would return `4`, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

Complete method `longestHeatWave`.

```
/**
 *  Returns the length of the longest heat wave found in temperatures, as described in
 *  part (b)
 *  Precondition: There is at least one heat wave in temperatures based on threshold.
 */
public int longestHeatWave(double threshold)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WeatherData

private ArrayList<Double> temperatures

public void cleanData(double lower, double upper)
public int longestHeatWave(double threshold)
```

3. Many encoded strings contain *delimiters*. A delimiter is a non-empty string that acts as a boundary between different parts of a larger string. The delimiters involved in this question occur in pairs that must be *balanced*, with each pair having an open delimiter and a close delimiter. There will be only one type of delimiter for each string. The following are examples of delimiters.

   Example 1
   Expressions in mathematics use open parentheses `"("` and close parentheses `")"` as delimiters. For each open parenthesis, there must be a matching close parenthesis.

`(x + y) * 5`	is a valid mathematical expression.
`(x + (y)`	is NOT a valid mathematical expression because there are more open delimiters than close delimiters.

   Example 2
   HTML uses `<B>` and `</B>` as delimiters. For each open delimiter `<B>`, there must be a matching close delimiter `</B>`.

`<B> Make this text bold </B>`	is valid HTML.
`<B> Make this text bold </UB>`	is NOT valid HTML because there is one open delimiter and no matching close delimiter.

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

    /** The open and close delimiters. */
    private String openDel;
    private String closeDel;

    /** Constructs a Delimiters object where open is the open delimiter and close is the
     *  close delimiter.
     *  Precondition: open and close are non-empty strings.
     */
    public Delimiters(String open, String close)
    {
       openDel = open;
       closeDel = close;
    }

    /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
    public ArrayList<String> getDelimitersList(String[] tokens)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
     *  Precondition: delimiters contains only valid open and close delimiters.
     */
    public boolean isBalanced(ArrayList<String> delimiters)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

`openDel:` `"("`

`closeDel:` `")"`

`tokens:`

`"("`	`"x + y"`	`")"`	`" * 5"`

`ArrayList` of delimiters:

`"("`	`")"`

Example 2

`openDel:` `"<q>"`

`closeDel:` `"</q>"`

`tokens:`

`"<q>"`	`"yy"`	`"</q>"`	`"zz"`	`"</q>"`

`ArrayList` of delimiters:

`"<q>"`	`"</q>"`	`"</q>"`

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

(b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1
The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

"^{"	"^{"	"}"	"^{"	"}"	"}"

Example 2
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"^{"	"}"	"</sup>"	"<sup>"

↑

When starting from the left, at this point, condition 1 is violated.

Example 3
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"</sup>"

↑

At this point, condition 1 is violated.

Example 4
The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.

"<sup>"	"^{"	"}"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
```

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }
```

(a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects.

A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects `(words[i], words[j])` for every `i` and `j`, where $0 \le i < j <$ `words.length`. Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 * Precondition:
```

(b) Write the `WordPairList` method `numMatches`. This method returns the number of `WordPair` objects in `allPairs` for which the two strings match.

For example, the following code segment creates a `WordPairList` object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return `2`.

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete method `numMatches` below.

```
/** Returns the number of matches as described in part (b).
 */
```

1. This question involves identifying and processing the digits of a non-negative integer. The declaration of the `Digits` class is shown below. You will write the constructor and one method for the `Digits` class.

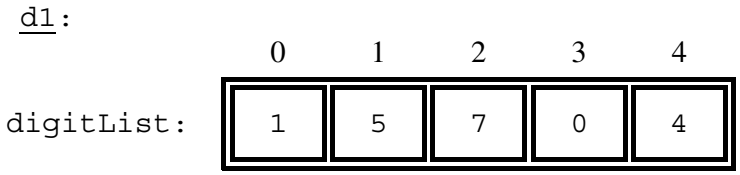
```
public class Digits
{
    /** The list of digits from the number used to construct this object.
     *  The digits appear in the list in the same order in which they appear in the original number.
     */
    private ArrayList<Integer> digitList;

    /** Constructs a Digits object that represents num.
     *  Precondition: num >= 0
     */
    public Digits(int num)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the digits in this Digits object are in strictly increasing order;
     *          false otherwise.
     */
    public boolean isStrictlyIncreasing()
    {  /* to be implemented in part (b) */  }
}
```

Part (a) begins on page 4.

(a) Write the constructor for the `Digits` class. The constructor initializes and fills `digitList` with the digits from the non-negative integer `num`. The elements in `digitList` must be `Integer` objects representing single digits, and appear in the same order as the digits in `num`. Each of the following examples shows the declaration of a `Digits` object and the contents of `digitList` as initialized by the constructor.

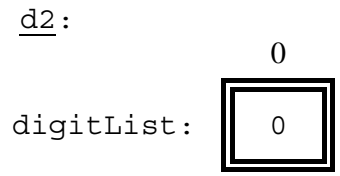
Example 1

```
Digits d1 = new Digits(15704);
```



Example 2

```
Digits d2 = new Digits(0);
```



WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete the `Digits` constructor below.

```
/** Constructs a Digits object that represents num.
 *  Precondition: num >= 0
 */
public Digits(int num)
```

Part (b) begins on page 6.

(b) Write the `Digits` method `isStrictlyIncreasing`. The method returns `true` if the elements of `digitList` appear in strictly increasing order; otherwise, it returns `false`. A list is considered strictly increasing if each element after the first is greater than (but not equal to) the preceding element.

The following table shows the results of several calls to `isStrictlyIncreasing`.

Method call	Value returned
`new Digits(7).isStrictlyIncreasing()`	`true`
`new Digits(1356).isStrictlyIncreasing()`	`true`
`new Digits(1336).isStrictlyIncreasing()`	`false`
`new Digits(1536).isStrictlyIncreasing()`	`false`
`new Digits(65310).isStrictlyIncreasing()`	`false`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `isStrictlyIncreasing` below.

```
/** Returns true if the digits in this Digits object are in strictly increasing order;
 *          false otherwise.
 */
```

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

   When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width.* Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

   The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

   Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

   Total number of letters in words: 14
   Number of gaps between words: 3
   Basic gap width: 2
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	P			C	O	M	P			S	C	I			R	O	C	K	S

   Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

   Total number of letters in words: 15
   Number of gaps between words: 3
   Basic gap width: 1
   Leftover spaces: 2

   The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	R	E	E	N			E	G	G	S			A	N	D		H	A	M

   Example 3: `wordList: ["BEACH", "BALL"]`

   Total number of letters in words: 9
   Number of gaps between words: 1
   Basic gap width: 11
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	E	A	C	H												B	A	L	L

   You will implement three `static` methods in a class named `StringFormatter` that is not shown.

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns `7`. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

Part (b) begins on page 20.

(b) Write the `StringFormatter` method `basicGapWidth`, which returns the basic gap width as defined earlier.

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
```

Part (c) begins on page 22.

(c) Write the `StringFormatter` method `format`, which returns the formatted string as defined earlier. The `StringFormatter` class also contains a method called `leftoverSpaces`, which has already been implemented. This method returns the number of leftover spaces as defined earlier and is shown below.

```
/** Returns the number of leftover spaces when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
{  /* implementation not shown */  }
```

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

STOP

END OF EXAM

3. A two-dimensional array of integers in which most elements are zero is called a *sparse array*. Because most elements have a value of zero, memory can be saved by storing only the non-zero values along with their row and column indexes. The following complete `SparseArrayEntry` class is used to represent non-zero elements in a sparse array. A `SparseArrayEntry` object cannot be modified after it has been constructed.

```
public class SparseArrayEntry
{
  /** The row index and column index for this entry in the sparse array */
  private int row;
  private int col;

  /** The value of this entry in the sparse array */
  private int value;

  /** Constructs a SparseArrayEntry object that represents a sparse array element
   *  with row index r and column index c, containing value v.
   */
  public SparseArrayEntry(int r, int c, int v)
  {
    row = r;
    col = c;
    value = v;
  }

  /** Returns the row index of this sparse array element. */
  public int getRow()
  {  return row;  }

  /** Returns the column index of this sparse array element. */
  public int getCol()
  {  return col;  }

  /** Returns the value of this sparse array element. */
  public int getValue()
  {  return value;  }
}
```

The `SparseArray` class represents a sparse array. It contains a list of `SparseArrayEntry` objects, each of which represents one of the non-zero elements in the array. The entries representing the non-zero elements are stored in the list in no particular order. Each non-zero element is represented by exactly one entry in the list.

```
public class SparseArray
{
  /** The number of rows and columns in the sparse array. */
  private int numRows;
  private int numCols;

  /** The list of entries representing the non-zero elements of the sparse array. Entries are stored in the
   *  list in no particular order. Each non-zero element is represented by exactly one entry in the list.
   */
  private List<SparseArrayEntry> entries;

  /** Constructs an empty SparseArray. */
  public SparseArray()
  {  entries = new ArrayList<SparseArrayEntry>();  }

  /** Returns the number of rows in the sparse array. */
  public int getNumRows()
  {  return numRows;  }

  /** Returns the number of columns in the sparse array. */
  public int getNumCols()
  {  return numCols;  }

  /** Returns the value of the element at row index row and column index col in the sparse array.
   *  Precondition: 0 ≤ row < getNumRows()
   *                0 ≤ col < getNumCols()
   */
  public int getValueAt(int row, int col)
  {  /* to be implemented in part (a) */  }

  /** Removes the column col from the sparse array.
   *  Precondition: 0 ≤ col < getNumCols()
   */
  public void removeColumn(int col)
  {  /* to be implemented in part (b) */  }

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

The following table shows an example of a two-dimensional sparse array. Empty cells in the table indicate zero values.

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The sample array can be represented by a `SparseArray` object, `sparse`, with the following instance variable values. The items in `entries` are in no particular order; one possible ordering is shown below.

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

(a) Write the `SparseArray` method `getValueAt`. The method returns the value of the sparse array element at a given row and column in the sparse array. If the list `entries` contains an entry with the specified row and column, the value associated with the entry is returned. If there is no entry in `entries` corresponding to the specified row and column, 0 is returned.

In the example above, the call `sparse.getValueAt(3, 1)` would return -9, and `sparse.getValueAt(3, 3)` would return 0.

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 12.

Complete method `getValueAt` below.

```
/** Returns the value of the element at row index row and column index col in the sparse array.
 *  Precondition: 0 ≤ row < getNumRows()
 *                0 ≤ col < getNumCols()
 */
public int getValueAt(int row, int col)
```

Part (b) begins on page 13.

(b) Write the `SparseArray` method `removeColumn`. After removing a specified column from a sparse array:

- All entries in the list `entries` with column indexes matching `col` are removed from the list.
- All entries in the list `entries` with column indexes greater than `col` are replaced by entries with column indexes that are decremented by one (moved one column to the left).
- The number of columns in the sparse array is adjusted to reflect the column removed.

The sample object `sparse` from the beginning of the question is repeated for your convenience.

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The shaded entries in `entries`, below, correspond to the shaded column above.

```
numRows: 6

numCols: 5

entries:
```

row: 1 col: 4 value: 4	row: 2 col: 0 value: 1	row: 3 col: 1 value: -9	row: 1 col: 1 value: 5

When `sparse` has the state shown above, the call `sparse.removeColumn(1)` could result in `sparse` having the following values in its instance variables (since `entries` is in no particular order, it would be equally valid to reverse the order of its two items). The shaded areas below show the changes.

```
numRows: 6

numCols: 4

entries:
```

row: 1 col: 3 value: 4	row: 2 col: 0 value: 1

Class information repeated from the beginning of the question

```
public class SparseArrayEntry

public SparseArrayEntry(int r, int c, int v)
public int getRow()
public int getCol()
public int getValue()

public class SparseArray

private int numRows
private int numCols
private List<SparseArrayEntry> entries
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int col)
public void removeColumn(int col)
```

Complete method `removeColumn` below.

```
/** Removes the column col from the sparse array.
 *  Precondition: 0 ≤ col < getNumCols()
 */
public void removeColumn(int col)
```

4. 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.

Part (b) begins on page 17.

(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.

Part (c) begins on page 18.

(c) The `MultipleGroups` class (not shown) represents a collection of `NumberGroup` objects and is a `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)
```

STOP

END OF EXAM

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the appendices 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 may not receive full credit.

1. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.

   (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.

   - The scrambling process begins at the first letter of the word and continues from left to right.
   - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
   - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

   The following table shows several examples of words and their scrambled versions.

`word`	Result returned by `scrambleWord(word)`
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    - the word was scrambled, beginning at the first letter and continuing from left to right
 *    - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    - letters were swapped at most once
 */
public static String scrambleWord(String word)
```

Part (b) begins on page 4.

(b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
`wordList`	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
`wordList`	"TNA"	"BARCADABARA"	"PAPLE"

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    - all words unchanged by scrambling have been removed from wordList
 *    - each of the remaining words has been replaced by its scrambled version
 *    - the relative ordering of the entries in wordList is the same as it was
 *          before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```

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

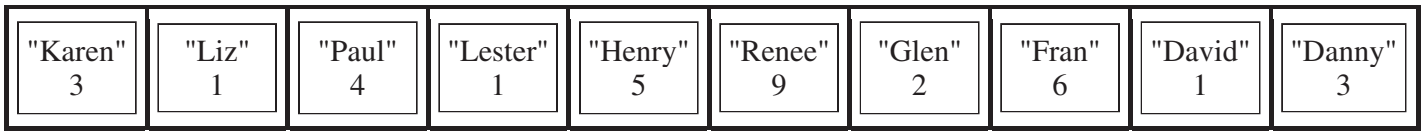
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** 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<Student> roster` contains references to `Student` objects in the following order.



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`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

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)
```

Part (b) begins on page 10.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.10 Implementing ArrayList Algorithms

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS array 数组 • arraylist 动态数组

Objective

Build the skills to answer exam questions on **implementing ArrayList algorithms**.

You must be able to:

- compute a sum, average, minimum, or maximum over an **ArrayList**
- **count** or **filter** elements that meet a condition
- insert or delete elements

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Summarizing an ArrayList

Just like an array, but using `get` and `size`:

```
int sum = 0;
for (int x : list) sum += x;
```

■ Counting, inserting, deleting

Use an `if` inside the loop to **count** matching elements. Use `add(index, value)` to **insert** and `remove(index)` to **delete**.

2 Practice

2.1 Describe how to sum the elements of an ArrayList. [1]

2.2 For an ArrayList holding `[3, 6, 9]`, find the sum. [1]

2.3 Name the method used to remove an element at a given index. [1]

3 Exam-style questions

3.1 To count elements meeting a condition in an ArrayList, you use a loop with [1]

- **A** no condition
- **B** an `if`
- **C** a cast
- **D** a constructor

3.2 `list.remove(0)` [1]

- **A** adds an element
- **B** removes the first element
- **C** returns the size
- **D** sorts the list

3.3 An ArrayList `list` holds `[10, 20, 30]`.

(a) Find the sum of its elements. [1]

(b) State `list.size()`. [1]

(c) Name the method that adds `40` to the end. [1]

Solutions

2.1 start a total at 0 and add each element as you loop through the list.

2.2 $3 + 6 + 9 = 18$.

2.3 `remove`.

3.1 B.

3.2 B.

3.3 (a) 60. (b) `3`. (c) `add`.

3. The `CourseRecord` class is used to store information about a student enrolled in a course. A partial definition of the `CourseRecord` class is shown.

```
public class CourseRecord
{
   /**
    *  Returns a unique ID for the student associated with this
    *  CourseRecord
    */
   public String getStudentID()
   { /* implementation not shown */ }

   /**
    *  Returns a nonnegative integer representing the number of times the
    *  student associated with this CourseRecord has been absent in the
    *  course
    */
   public int getAbsences()
   { /* implementation not shown */ }

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

The `Attendance` class maintains two `ArrayLists` of `CourseRecord` objects, named `historyList` and `mathList`. A partial definition of the `Attendance` class is shown.

```
public class Attendance
{
   /* The students enrolled in a history course */
   private ArrayList<CourseRecord> historyList;

   /* The students enrolled in a math course */
   private ArrayList<CourseRecord> mathList;

   /**
    *  Returns the number of students who are enrolled in both
    *  the history course and the math course but have more
    *  absences in the history course than the math course
    *  Preconditions:
    *        No student ID appears multiple times in historyList.
    *        No student ID appears multiple times in mathList.
    *        historyList and mathList do not contain any null elements.
    *  Postcondition:
    *        historyList and mathList are unchanged.
    */
   public int moreHistoryThanMathAbsences()
   { /* to be implemented */ }

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

Write the `Attendance` method `moreHistoryThanMathAbsences`. The method should return the number of students who are enrolled in both the history course and the math course but have more absences in the history course than the math course.

For example, suppose `historyList` and `mathList` have the following contents.

`historyList`:

Student ID	"rc29"	"br98"	"dr03"	"ot32"	"sq98"	"ry00"
Number of Absences	1	1	2	2	3	1

`mathList`:

Student ID	"fr27"	"sq98"	"dr03"	"dk12"	"ot32"	"js33"	"ry00"
Number of Absences	2	1	2	1	1	0	3

In this example, there are four student IDs (`"dr03"`, `"ot32"`, `"sq98"`, and `"ry00"`) that appear in both `historyList` and `mathList`. Of these, only `"ot32"` and `"sq98"` have more absences in the history course than the math course, so the `moreHistoryThanMathAbsences` method should return `2`.

Complete method `moreHistoryThanMathAbsences`.

```
/**
 *  Returns the number of students who are enrolled in both
 *  the history course and the math course but have more
 *  absences in the history course than the math course
 *  Preconditions:
 *        No student ID appears multiple times in historyList.
 *        No student ID appears multiple times in mathList.
 *        historyList and mathList do not contain any null elements.
 *  Postcondition:
 *        historyList and mathList are unchanged.
 */
public int moreHistoryThanMathAbsences()
```

3. This question involves pairing competitors in a tournament into one-on-one matches for one round of the tournament. For example, in a chess tournament, the competitors are the individual chess players. A game of chess involving two players is a match. The winner of each match goes on to a match in the next round of the tournament. Since half of the players are eliminated in each round of the tournament, there is eventually a final round consisting of one match and two competitors. The winner of that match is considered the winner of the tournament.

Competitors, matches, and rounds of the tournament are represented by the `Competitor`, `Match`, and `Round` classes. You will write the constructor and one method of the `Round` class.

```
/** A single competitor in the tournament */
public class Competitor
{
  /** The competitor's name and rank */
  private String name;
  private int rank;

  /**
   * Assigns n to name and initialRank to rank
   * Precondition: initialRank >= 1
   */
  public Competitor(String n, int initialRank)
  { /* implementation not shown */ }

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

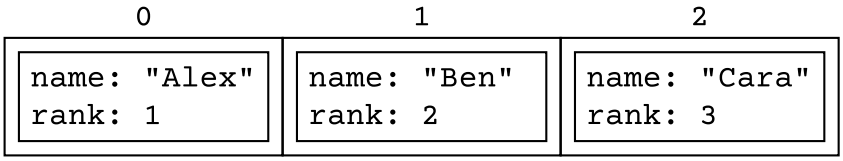
```
/** A match between two competitors */
public class Match
{
  public Match(Competitor one, Competitor two)
  { /* implementation not shown */ }

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


/** A single round of the tournament */
public class Round
{
  /** The list of competitors participating in this round */
  private ArrayList<Competitor> competitorList;

  /** Initializes competitorList, as described in part (a) */
  public Round(String[] names)
  { /* to be implemented in part (a) */ }

  /**
   * Creates an ArrayList of Match objects for the next round
   * of the tournament, as described in part (b)
   * Preconditions: competitorList contains at least one element.
   *                competitorList is ordered from best to worst rank.
   * Postcondition: competitorList is unchanged.
   */
  public ArrayList<Match> buildMatches()
  { /* to be implemented in part (b) */ }

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

A. Write the constructor for the `Round` class. The constructor should initialize `competitorList` to contain one `Competitor` object for each name in the `String[] names`. The order in which `Competitor` objects appear in `competitorList` should be the same as the order in which they appear in `names`, and the rank of each competitor is based on the competitor's position in `names`. Names are listed in `names` in order from the best-ranked competitor with rank `1` to the worst-ranked competitor with rank *n*, where *n* is the number of elements in `names`.

For example, assume the following code segment is executed.

```
String[] players = {"Alex", "Ben", "Cara"};
Round r = new Round(players);
```

The following shows the contents of `competitorList` in `r` after the constructor has finished executing.



Complete the `Round` constructor.

```
/** Initializes competitorList, as described in part (a) */
public Round(String[] names)
```

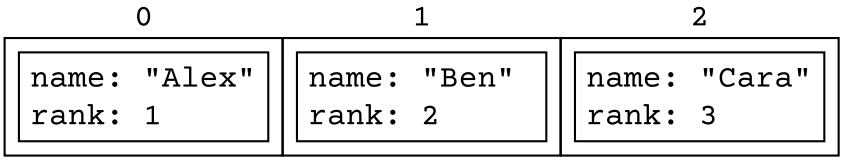
B. Write the `Round` method `buildMatches`. This method should return a new `ArrayList<Match>` object by pairing competitors from `competitorList` according to the following rules.

- If the number of competitors in `competitorList` is even, the best-ranked competitor is paired with the worst-ranked competitor, the second-best-ranked competitor is paired with the second-worst-ranked competitor, etc.
- If the number of competitors in `competitorList` is odd, the competitor with the best rank is ignored and the remaining competitors are paired according to the rule for an even number of competitors.

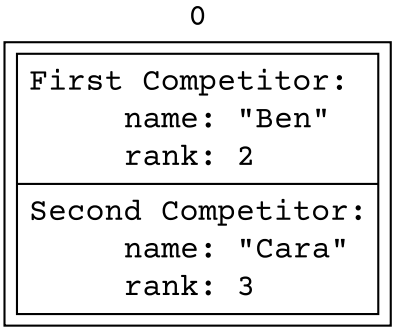
Each pair of competitors is used to create a `Match` object that should be added to the `ArrayList` to return. Competitors may appear in either order in a `Match` object, and matches may appear in any order in the returned `ArrayList`.

The following example shows the contents of `competitorList` in a `Round` object `r1` containing an odd number of competitors and the `ArrayList` of `Match` objects that should be returned by the call `r1.buildMatches()`.

`competitorList`:

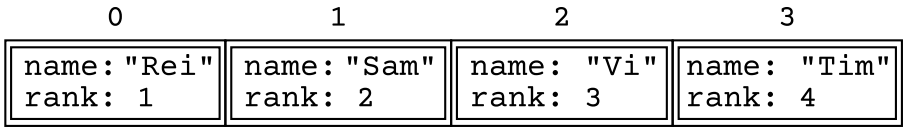


The `ArrayList` to be returned contains a single match between Ben and Cara, the second and third-ranked competitors:

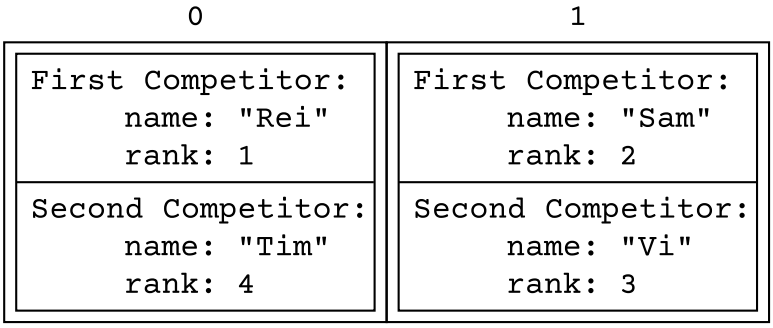


The next example shows the contents of `competitorList` in a `Round` object `r2` containing an even number of competitors and the `ArrayList` of `Match` objects that should be returned by the call `r2.buildMatches()`.

`competitorList`:



The `ArrayList` to be returned contains two matches: one match between the first- and last-ranked competitors Rei and Tim, and a second match between the second- and third-ranked competitors Sam and Vi:



Complete the `buildMatches` method.

```
/**
 * Creates an ArrayList of Match objects for the next round
 * of the tournament, as described in part (b)
 * Preconditions: competitorList contains at least one element.
 *                competitorList is ordered from best to worst rank.
 * Postcondition: competitorList is unchanged.
 */
public ArrayList<Match> buildMatches()
```

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
   /** Initialized in the constructor and contains no null elements */
   private ArrayList<String> wordList;

   /**
    *  Returns true if each element of wordList (except the first) contains the previous
    *  element as a substring and returns false otherwise, as described in part (a)
    *  Precondition: wordList contains at least two elements.
    *  Postcondition: wordList is unchanged.
    */
   public boolean isWordChain()
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns an ArrayList<String> based on strings from wordList that start
    *  with target, as described in part (b). Each element of the returned ArrayList has had
    *  the initial occurrence of target removed.
    *  Postconditions: wordList is unchanged.
    *       Items appear in the returned list in the same order as they appear in wordList.
    */
   public ArrayList<String> createList(String target)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

`wordList`	`isWordChain` **Return Value**	**Explanation**
`["an", "band", "band", "abandon"]`	`true`	Each element contains the previous element as a substring.
`["to", "too", "stool", "tools"]`	`false`	`"tools"` does not contain the substring `"stool"`.

Complete the `isWordChain` method.

```
/**
 *  Returns true if each element of wordList (except the first) contains the previous
 *  element as a substring and returns false otherwise, as described in part (a)
 *  Precondition: wordList contains at least two elements.
 *  Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	`ArrayList` **Returned by** `createList`	**Explanation**
`createList("cat")`	`["ch", "chacat", ""]`	Only `"catch"`, `"catchacat"`, and `"cat"` begin with `"cat"`.
`createList("catch")`	`["", "acat"]`	Only `"catch"` and `"catchacat"` begin with `"catch"`.
`createList("dog")`	`[]`	None of the words in `wordList` begin with `"dog"`.

Complete the `createList` method.

```
/**
 *  Returns an ArrayList<String> based on strings from wordList that start
 *  with target, as described in part (b). Each element of the returned ArrayList has had
 *  the initial occurrence of target removed.
 *  Postconditions: wordList is unchanged.
 *      Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

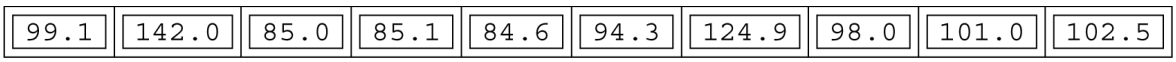
```
public class WordChecker

private ArrayList<String> wordList

public boolean isWordChain()
public ArrayList<String> createList(String target)
```

3. This question involves the analysis of weather data. The following `WeatherData` class has an instance variable, `temperatures`, which contains the daily high temperatures recorded on consecutive days at a particular location. The class also contains methods used to analyze that data. You will write two methods of the `WeatherData` class.

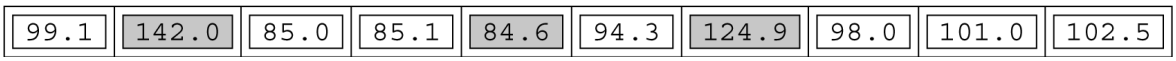
```
public class WeatherData
{
    /** Guaranteed not to be null and to contain only non-null entries */
    private ArrayList<Double> temperatures;

    /**
     *  Cleans the data by removing from temperatures all values that are less than
     *  lower and all values that are greater than upper, as described in part (a)
     */
    public void cleanData(double lower, double upper)
    {  /* to be implemented in part (a) */  }

    /**
     *  Returns the length of the longest heat wave found in temperatures, as described in
     *  part (b)
     *  Precondition: There is at least one heat wave in temperatures based on threshold.
     */
    public int longestHeatWave(double threshold)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `cleanData` method, which modifies the `temperatures` instance variable by removing all values that are less than the `lower` parameter and all values that are greater than the `upper` parameter. The order of the remaining values in `temperatures` must be maintained.

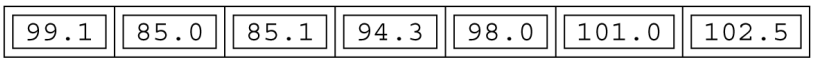
For example, consider a `WeatherData` object for which `temperatures` contains the following.



The three shaded values shown would be removed by the method call `cleanData(85.0, 120.0)`.



The following shows the contents of `temperatures` after the three shaded values are removed as a result of the method call `cleanData(85.0, 120.0)`.



Complete method `cleanData`.

```
/**
 *  Cleans the data by removing from temperatures all values that are less than
 *  lower and all values that are greater than upper, as described in part (a)
 */
public void cleanData(double lower, double upper)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `longestHeatWave` method, which returns the length of the longest heat wave found in the `temperatures` instance variable. A heat wave is a sequence of two or more consecutive days with a daily high temperature greater than the parameter `threshold`. The `temperatures` instance variable is guaranteed to contain at least one heat wave based on the `threshold` parameter.

For example, consider the following contents of `temperatures`.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

In the following sample contents of `temperatures`, all heat waves based on the `threshold` temperature of `100.5` are shaded. The method call `longestHeatWave(100.5)` would return `3`, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

In the following sample contents of `temperatures`, all heat waves based on the `threshold` temperature of `95.2` are shaded. The method call `longestHeatWave(95.2)` would return `4`, which is the length of the longest heat wave.

100.5	98.5	102.0	103.9	87.5	105.2	90.3	94.8	109.1	102.1	107.4	93.2

Complete method `longestHeatWave`.

```
/**
 *  Returns the length of the longest heat wave found in temperatures, as described in
 *  part (b)
 *  Precondition: There is at least one heat wave in temperatures based on threshold.
 */
public int longestHeatWave(double threshold)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WeatherData

private ArrayList<Double> temperatures

public void cleanData(double lower, double upper)
public int longestHeatWave(double threshold)
```

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (`"."`), exclamation point (`"!"`), or letter, or is a `String` of length `0` if the user did not enter a comment.

```
public class Review
{
   private int rating;
   private String comment;

   /** Precondition: r >= 0
    *      c is not null.
    */
   public Review(int r, String c)
   {
      rating = r;
      comment = c;
   }

   public int getRating()
   {
      return rating;
   }

   public String getComment()
   {
      return comment;
   }

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

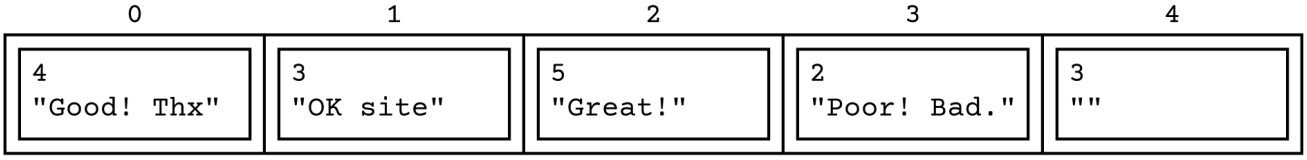
GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    {  /* implementation not shown */  }

    /** Returns a double representing the average rating of all the Review objects to be
     *  analyzed, as described in part (a)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     */
    public double getAverageRating()
    {  /* to be implemented in part (a) */  }

    /** Returns an ArrayList of String objects containing formatted versions of
     *  selected user comments, as described in part (b)
     *  Precondition: allReviews contains at least one Review.
     *      No element of allReviews is null.
     *  Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return `3.4` if `allReviews` contained the following `Review` objects.



Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 *  analyzed, as described in part (a)
 *  Precondition: allReviews contains at least one Review.
 *      No element of allReviews is null.
 */
public double getAverageRating()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

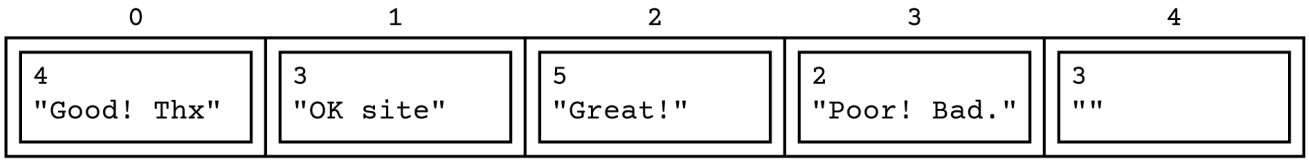
```
public class Review

private int rating
private String comment

public Review(int r, String c)
public int getRating()
public String getComment()

public class ReviewAnalysis

private Review[] allReviews

public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

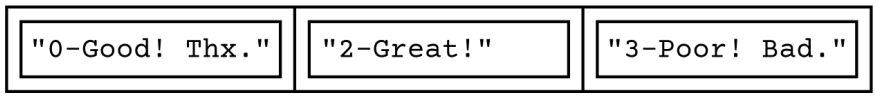
(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (`"-"`).
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).



The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index `1` and index `4` in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.



Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 *  selected user comments, as described in part (b)
 *  Precondition: allReviews contains at least one Review.
 *       No element of allReviews is null.
 *  Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

3. A high school club maintains information about its members in a `MemberInfo` object. A `MemberInfo` object stores a club member's name, year of graduation, and whether or not the club member is in *good standing*. A member who is in good standing has fulfilled all the responsibilities of club membership.

   A partial declaration of the `MemberInfo` class is shown below.

```
public class MemberInfo
{
   /** Constructs a MemberInfo object for the club member with name name,
    *  graduation year gradYear, and standing hasGoodStanding.
    */
   public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
   {  /* implementation not shown */  }

   /** Returns the graduation year of the club member. */
   public int getGradYear()
   {  /* implementation not shown */  }

   /** Returns true if the member is in good standing and false otherwise. */
   public boolean inGoodStanding()
   {  /* implementation not shown */  }

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

   The `ClubMembers` class maintains a list of current club members. The declaration of the `ClubMembers` class is shown below.

```
public class ClubMembers
{
   private ArrayList<MemberInfo> memberList;

   /** Adds new club members to memberList, as described in part (a).
    *  Precondition: names is a non-empty array.
    */
   public void addMembers(String[] names, int gradYear)
   {  /* to be implemented in part (a) */  }

   /** Removes members who have graduated and returns a list of members who have graduated
    *  and are in good standing, as described in part (b).
    */
   public ArrayList<MemberInfo> removeMembers(int year)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `ClubMembers` method `addMembers`, which takes two parameters. The first parameter is a `String` array containing the names of new club members to be added. The second parameter is the graduation year of all the new club members. The method adds the new members to the `memberList` instance variable. The names can be added in any order. All members added are initially in good standing and share the same graduation year, `gradYear`.

Complete the `addMembers` method.

```
/** Adds new club members to memberList, as described in part (a).
 *  Precondition: names is a non-empty array.
 */
public void addMembers(String[] names, int gradYear)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `ClubMembers` method `removeMembers`, which takes the following actions.

- Returns a list of all students who have graduated and are in good standing. A member has graduated if the member's graduation year is less than or equal to the method's `year` parameter. If no members meet these criteria, an empty list is returned.
- Removes from `memberList` all members who have graduated, regardless of whether or not they are in good standing.

The following example illustrates the results of a call to `removeMembers`.

The `ArrayList memberList` before the method call `removeMembers(2018)`:

"SMITH, JANE"	"FOX, STEVE"	"XIN, MICHAEL"	"GARCIA, MARIA"
2019	2018	2017	2020
false	true	false	true

The `ArrayList memberList` after the method call `removeMembers(2018)`:

"SMITH, JANE"	"GARCIA, MARIA"
2019	2020
false	true

The `ArrayList` returned by the method call `removeMembers(2018)`:

"FOX, STEVE"
2018
true

Complete the `removeMembers` method.

```
/** Removes members who have graduated and returns a list of members who have graduated and are
 *  in good standing, as described in part (b).
 */
public ArrayList<MemberInfo> removeMembers(int year)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class MemberInfo

public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
public int getGradYear()
public boolean inGoodStanding()

public class ClubMembers

private ArrayList<MemberInfo> memberList

public void addMembers(String[] names, int gradYear)
public ArrayList<MemberInfo> removeMembers(int year)
```

3. Many encoded strings contain *delimiters*. A delimiter is a non-empty string that acts as a boundary between different parts of a larger string. The delimiters involved in this question occur in pairs that must be *balanced*, with each pair having an open delimiter and a close delimiter. There will be only one type of delimiter for each string. The following are examples of delimiters.

   Example 1
   Expressions in mathematics use open parentheses `"("` and close parentheses `")"` as delimiters. For each open parenthesis, there must be a matching close parenthesis.

`(x + y) * 5`	is a valid mathematical expression.
`(x + (y)`	is NOT a valid mathematical expression because there are more open delimiters than close delimiters.

   Example 2
   HTML uses `<B>` and `</B>` as delimiters. For each open delimiter `<B>`, there must be a matching close delimiter `</B>`.

`<B> Make this text bold </B>`	is valid HTML.
`<B> Make this text bold </UB>`	is NOT valid HTML because there is one open delimiter and no matching close delimiter.

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

   /** The open and close delimiters. */
   private String openDel;
   private String closeDel;

   /** Constructs a Delimiters object where open is the open delimiter and close is the
    *  close delimiter.
    *  Precondition: open and close are non-empty strings.
    */
   public Delimiters(String open, String close)
   {
      openDel = open;
      closeDel = close;
   }

   /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
   public ArrayList<String> getDelimitersList(String[] tokens)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
    *  Precondition: delimiters contains only valid open and close delimiters.
    */
   public boolean isBalanced(ArrayList<String> delimiters)
   {  /* to be implemented in part (b) */  }

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

}
```

(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

`openDel:` `"("`

`closeDel:` `")"`

`tokens:`

`"("`	`"x + y"`	`")"`	`" * 5"`

`ArrayList` of delimiters:

`"("`	`")"`

Example 2

`openDel:` `"<q>"`

`closeDel:` `"</q>"`

`tokens:`

`"<q>"`	`"yy"`	`"</q>"`	`"zz"`	`"</q>"`

`ArrayList` of delimiters:

`"<q>"`	`"</q>"`	`"</q>"`

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

(b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1
The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

"^{"	"^{"	"}"	"^{"	"}"	"}"

Example 2
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"^{"	"}"	"</sup>"	"<sup>"

↑
When starting from the left, at this point, condition 1 is violated.

Example 3
The following example shows an `ArrayList` for which `isBalanced` returns `false`.

"</sup>"

↑
At this point, condition 1 is violated.

Example 4
The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.

"<sup>"	"^{"	"}"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
```

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }
```

(a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects.

A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects `(words[i], words[j])` for every `i` and `j`, where $0 \le i < j <$ `words.length`. Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 * Precondition:
```

(b) Write the `WordPairList` method `numMatches`. This method returns the number of `WordPair` objects in `allPairs` for which the two strings match.

For example, the following code segment creates a `WordPairList` object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return `2`.

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete method `numMatches` below.

```
/** Returns the number of matches as described in part (b).
 */
```

1. This question involves identifying and processing the digits of a non-negative integer. The declaration of the `Digits` class is shown below. You will write the constructor and one method for the `Digits` class.

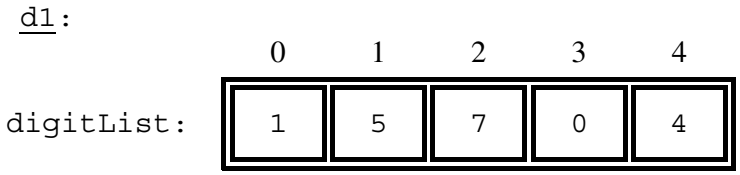
```
public class Digits
{
    /** The list of digits from the number used to construct this object.
     *  The digits appear in the list in the same order in which they appear in the original number.
     */
    private ArrayList<Integer> digitList;

    /** Constructs a Digits object that represents num.
     *  Precondition: num >= 0
     */
    public Digits(int num)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the digits in this Digits object are in strictly increasing order;
     *          false otherwise.
     */
    public boolean isStrictlyIncreasing()
    {  /* to be implemented in part (b) */  }
}
```

Part (a) begins on page 4.

(a) Write the constructor for the `Digits` class. The constructor initializes and fills `digitList` with the digits from the non-negative integer `num`. The elements in `digitList` must be `Integer` objects representing single digits, and appear in the same order as the digits in `num`. Each of the following examples shows the declaration of a `Digits` object and the contents of `digitList` as initialized by the constructor.

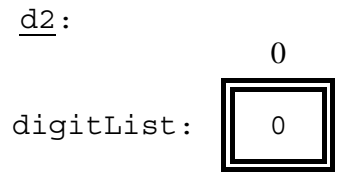
Example 1

```
Digits d1 = new Digits(15704);
```



Example 2

```
Digits d2 = new Digits(0);
```



WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete the `Digits` constructor below.

```
/** Constructs a Digits object that represents num.
 *  Precondition: num >= 0
 */
public Digits(int num)
```

Part (b) begins on page 6.

(b) Write the `Digits` method `isStrictlyIncreasing`. The method returns `true` if the elements of `digitList` appear in strictly increasing order; otherwise, it returns `false`. A list is considered strictly increasing if each element after the first is greater than (but not equal to) the preceding element.

The following table shows the results of several calls to `isStrictlyIncreasing`.

Method call	Value returned
`new Digits(7).isStrictlyIncreasing()`	`true`
`new Digits(1356).isStrictlyIncreasing()`	`true`
`new Digits(1336).isStrictlyIncreasing()`	`false`
`new Digits(1536).isStrictlyIncreasing()`	`false`
`new Digits(65310).isStrictlyIncreasing()`	`false`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `isStrictlyIncreasing` below.

```
/** Returns true if the digits in this Digits object are in strictly increasing order;
 *          false otherwise.
 */
```

2. This question involves two classes that are used to process log messages. A list of sample log messages is given below.

```
CLIENT3:security alert – repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

Log messages have the format *machineId*:*description*, where *machineId* identifies the computer and *description* describes the event being logged. Exactly one colon (":") appears in a log message. There are no blanks either immediately before or immediately after the colon.

The following `LogMessage` class is used to represent a log message.

```
public class LogMessage
{
   private String machineId;
   private String description;

   /** Precondition: message is a valid log message. */
   public LogMessage(String message)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the description in this log message properly contains keyword;
    *          false otherwise.
    */
   public boolean containsWord(String keyword)
   {  /* to be implemented in part (b) */  }

   public String getMachineId()
   {  return machineId;  }

   public String getDescription()
   {  return description;  }

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

(a) Write the constructor for the `LogMessage` class. It must initialize the private data of the object so that `getMachineId` returns the *machineId* part of the message and `getDescription` returns the *description* part of the message.

Complete the `LogMessage` constructor below.

```
/** Precondition: message is a valid log message. */
public LogMessage(String message)
```

Part (b) begins on page 8.

(b) Write the `LogMessage` method `containsWord`, which returns `true` if the description in the log message *properly contains* a given keyword and returns `false` otherwise.

A description *properly contains* a keyword if all three of the following conditions are true.

- the keyword is a substring of the description;
- the keyword is either at the beginning of the description or it is immediately preceded by a space;
- the keyword is either at the end of the description or it is immediately followed by a space.

The following tables show several examples. The descriptions in the left table properly contain the keyword `"disk"`. The descriptions in the right table do not properly contain the keyword `"disk"`.

Descriptions that properly contain `"disk"`

`"disk"`
`"error on disk"`
`"error on /dev/disk disk"`
`"error on disk DSK1"`

Descriptions that do not properly contain `"disk"`

`"DISK"`
`"error on disk3"`
`"error on /dev/disk"`
`"diskette"`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that the `LogMessage` constructor works as specified, regardless of what you wrote in part (a).

Complete method `containsWord` below.

```
/** Returns true if the description in this log message properly contains keyword;
 *         false otherwise.
 */
public boolean containsWord(String keyword)
```

Part (c) begins on page 10.

(c) The `SystemLog` class represents a list of `LogMessage` objects and provides a method that removes and returns a list of all log messages (if any) that properly contain a given keyword. The messages in the returned list appear in the same order in which they originally appeared in the system log. If no message properly contains the keyword, an empty list is returned. The declaration of the `SystemLog` class is shown below.

```
public class SystemLog
{
    /** Contains all the entries in this system log.
     *  Guaranteed not to be null and to contain only non-null entries.
     */
    private List<LogMessage> messageList;

    /** Removes from the system log all entries whose descriptions properly contain keyword,
     *  and returns a list (possibly empty) containing the removed entries.
     *  Postcondition:
     *    – Entries in the returned list properly contain keyword and
     *      are in the order in which they appeared in the system log.
     *    – The remaining entries in the system log do not properly contain keyword and
     *      are in their original order.
     *    – The returned list is empty if no messages properly contain keyword.
     */
    public List<LogMessage> removeMessages(String keyword)
    {  /* to be implemented in part (c) */  }

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

Write the `SystemLog` method `removeMessages`, which removes from the system log all entries whose descriptions properly contain `keyword` and returns a list of the removed entries in their original order. For example, assume that `theLog` is a `SystemLog` object initially containing six `LogMessage` objects representing the following list of log messages.

```
CLIENT3:security alert – repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

The call `theLog.removeMessages("disk")` would return a list containing the `LogMessage` objects representing the following log messages.

```
Webserver:disk offline
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
```

After the call, `theLog` would contain the following log messages.

```
CLIENT3:security alert – repeated login failures
SERVER1:file not found
Webserver:error on /dev/disk
```

Assume that the `LogMessage` class works as specified, regardless of what you wrote in parts (a) and (b). You must use `containsWord` appropriately to receive full credit.

Complete method `removeMessages` below.

```
/** Removes from the system log all entries whose descriptions properly contain keyword,
 *  and returns a list (possibly empty) containing the removed entries.
 *  Postcondition:
 *    – Entries in the returned list properly contain keyword and
 *      are in the order in which they appeared in the system log.
 *    – The remaining entries in the system log do not properly contain keyword and
 *      are in their original order.
 *    – The returned list is empty if no messages properly contain keyword.
 */
public List<LogMessage> removeMessages(String keyword)
```


3. A two-dimensional array of integers in which most elements are zero is called a *sparse array*. Because most elements have a value of zero, memory can be saved by storing only the non-zero values along with their row and column indexes. The following complete `SparseArrayEntry` class is used to represent non-zero elements in a sparse array. A `SparseArrayEntry` object cannot be modified after it has been constructed.

```
public class SparseArrayEntry
{
  /** The row index and column index for this entry in the sparse array */
  private int row;
  private int col;

  /** The value of this entry in the sparse array */
  private int value;

  /** Constructs a SparseArrayEntry object that represents a sparse array element
   *  with row index r and column index c, containing value v.
   */
  public SparseArrayEntry(int r, int c, int v)
  {
    row = r;
    col = c;
    value = v;
  }

  /** Returns the row index of this sparse array element. */
  public int getRow()
  {  return row;  }

  /** Returns the column index of this sparse array element. */
  public int getCol()
  {  return col;  }

  /** Returns the value of this sparse array element. */
  public int getValue()
  {  return value;  }
}
```

The `SparseArray` class represents a sparse array. It contains a list of `SparseArrayEntry` objects, each of which represents one of the non-zero elements in the array. The entries representing the non-zero elements are stored in the list in no particular order. Each non-zero element is represented by exactly one entry in the list.

```
public class SparseArray
{
  /** The number of rows and columns in the sparse array. */
  private int numRows;
  private int numCols;

  /** The list of entries representing the non-zero elements of the sparse array. Entries are stored in the
   *  list in no particular order. Each non-zero element is represented by exactly one entry in the list.
   */
  private List<SparseArrayEntry> entries;

  /** Constructs an empty SparseArray. */
  public SparseArray()
  {  entries = new ArrayList<SparseArrayEntry>();  }

  /** Returns the number of rows in the sparse array. */
  public int getNumRows()
  {  return numRows;  }

  /** Returns the number of columns in the sparse array. */
  public int getNumCols()
  {  return numCols;  }

  /** Returns the value of the element at row index row and column index col in the sparse array.
   *  Precondition: 0 ≤ row < getNumRows()
   *                0 ≤ col < getNumCols()
   */
  public int getValueAt(int row, int col)
  {  /* to be implemented in part (a) */  }

  /** Removes the column col from the sparse array.
   *  Precondition: 0 ≤ col < getNumCols()
   */
  public void removeColumn(int col)
  {  /* to be implemented in part (b) */  }

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

The following table shows an example of a two-dimensional sparse array. Empty cells in the table indicate zero values.

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The sample array can be represented by a `SparseArray` object, `sparse`, with the following instance variable values. The items in `entries` are in no particular order; one possible ordering is shown below.

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

(a) Write the `SparseArray` method `getValueAt`. The method returns the value of the sparse array element at a given row and column in the sparse array. If the list `entries` contains an entry with the specified row and column, the value associated with the entry is returned. If there is no entry in `entries` corresponding to the specified row and column, 0 is returned.

In the example above, the call `sparse.getValueAt(3, 1)` would return -9, and `sparse.getValueAt(3, 3)` would return 0.

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 12.

Complete method `getValueAt` below.

```
/** Returns the value of the element at row index row and column index col in the sparse array.
 *  Precondition: 0 ≤ row < getNumRows()
 *                0 ≤ col < getNumCols()
 */
public int getValueAt(int row, int col)
```

Part (b) begins on page 13.

(b) Write the `SparseArray` method `removeColumn`. After removing a specified column from a sparse array:

- All entries in the list `entries` with column indexes matching `col` are removed from the list.
- All entries in the list `entries` with column indexes greater than `col` are replaced by entries with column indexes that are decremented by one (moved one column to the left).
- The number of columns in the sparse array is adjusted to reflect the column removed.

The sample object `sparse` from the beginning of the question is repeated for your convenience.

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The shaded entries in `entries`, below, correspond to the shaded column above.

```
numRows: 6

numCols: 5

entries:
```

row: 1 col: 4 value: 4	row: 2 col: 0 value: 1	row: 3 col: 1 value: -9	row: 1 col: 1 value: 5

When `sparse` has the state shown above, the call `sparse.removeColumn(1)` could result in `sparse` having the following values in its instance variables (since `entries` is in no particular order, it would be equally valid to reverse the order of its two items). The shaded areas below show the changes.

```
numRows: 6

numCols: 4

entries:
```

row: 1 col: 3 value: 4	row: 2 col: 0 value: 1

Class information repeated from the beginning of the question

```
public class SparseArrayEntry

public SparseArrayEntry(int r, int c, int v)
public int getRow()
public int getCol()
public int getValue()

public class SparseArray

private int numRows
private int numCols
private List<SparseArrayEntry> entries
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int col)
public void removeColumn(int col)
```

Complete method `removeColumn` below.

```
/** Removes the column col from the sparse array.
 *  Precondition: 0 ≤ col < getNumCols()
 */
public void removeColumn(int col)
```

4. 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.

Part (b) begins on page 17.

(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.

Part (c) begins on page 18.

(c) The `MultipleGroups` class (not shown) represents a collection of `NumberGroup` objects and is a `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)
```

STOP

END OF EXAM

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.11 2D Array Creation and Access

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS two-dimensional array 二维数组 • array 数组

Objective

Build the skills to answer exam questions on **2D array creation and access**.

You must be able to:

- create a **two-dimensional array** 二维数组 arranged in rows and columns
- access an element using two indices, `grid[row][col]`, both from 0
- find the number of rows and columns

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Creating and accessing

```
int[][] grid = new int[3][4];   // 3 rows, 4 columns
grid[1][2] = 9;                 // row 1, column 2
```

Both indices start at 0.

■ Dimensions

- `grid.length` — the number of **rows**.
- `grid[0].length` — the number of **columns**.

2 Practice

2.1 State how to access the element in row 1, column 2 of `grid`. [1]

2.2 For `int[][] g = new int[3][5];`, state the number of rows and columns. [2]

2.3 State how to find the number of rows of a 2D array `g`. [1]

---

3 Exam-style questions

3.1 A 2D array element is accessed with [1]

- **A** `grid[row]`
- **B** `grid[row][col]`
- **C** `grid(row, col)`
- **D** `grid.get(row)`

3.2 `grid.length` gives the number of [1]

- **A** columns
- **B** rows
- **C** total elements
- **D** bytes

3.3 `int[][] g = new int[4][6];`.

(a) State the number of rows. [1]

(b) State the number of columns. [1]

(c) State how to access the element in row 0, column 0. [1]

Solutions

2.1 `grid[1][2]`.

2.2 3 rows and 5 columns.

2.3 `g.length`.

3.1 **B**.

3.2 **B**.

3.3 (a) `4`. (b) `6`. (c) `g[0][0]`.

4. This question involves reasoning about a number puzzle that is represented as a two-dimensional array of integers. Each element of the array initially contains a value between 1 and 9, inclusive. Solving the puzzle involves clearing pairs of array elements by setting them to 0. Two elements can be cleared if their values sum to 10 or if they have the same value. The puzzle is considered solved if all elements of the array are cleared.

You will write the constructor and one method of the `SumOrSameGame` class, which contains the methods that manipulate elements of the puzzle.

```
public class SumOrSameGame
{
  private int[][] puzzle;

  /**
   * Creates a two-dimensional array and fills it with random integers,
   * as described in part (a)
   * Precondition: numRows > 0; numCols > 0
   */
  public SumOrSameGame(int numRows, int numCols)
  { /* to be implemented in part (a) */ }

  /**
   * Identifies and clears an element of puzzle that can be paired with
   * the element at the given row and column, as described in part (b)
   * Preconditions: row and col are valid row and column indices in puzzle.
   *  The element at the given row and column is between 1 and 9, inclusive.
   */
  public boolean clearPair(int row, int col)
  { /* to be implemented in part (b) */ }

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

A. Write the constructor for the `SumOrSameGame` class. The constructor initializes the instance variable `puzzle` to be a two-dimensional integer array with the number of rows and columns specified by the parameters `numRows` and `numCols`, respectively. Array elements are initialized with random integers between `1` and `9`, inclusive, each with an equal chance of being assigned to each element of `puzzle`.

When an element of the two-dimensional array is accessed, the first index is used to specify the row and the second index is used to specify the column.

Complete the `SumOrSameGame` constructor.

```
/**
 * Creates a two-dimensional array and fills it with random integers,
 * as described in part (a)
 * Precondition: numRows > 0; numCols > 0
 */
public SumOrSameGame(int numRows, int numCols)
```

B. Write the `clearPair` method, which takes a valid row index and valid column index as its parameters. The array element specified by those indices, which has a value between `1` and `9`, inclusive, is compared to other array elements in `puzzle` in an attempt to pair it with another array element that meets both of the following conditions.

- The row index of the second element is greater than or equal to the parameter `row`.
- The two elements have equal values or have values that sum to 10.

If such an array element is found, both array elements of the pair are cleared (set to `0`) and the method returns `true`. If more than one such array element is found, any one of those identified array elements can be used to complete the pair and can be cleared. If no such array element is found, no changes are made to `puzzle` and the method returns `false`.

The following table shows the possible results of several calls to `clearPair`.

`puzzle` before call to `clearPair`	Method Call	`puzzle` after call to `clearPair`	Return Value	Explanation

`0 7 9 0`
`7 4 1 6`
`8 4 1 8`
	`clearPair(0, 1)`	`0 0 9 0`
		`0 4 1 6`
		`8 4 1 8`
			`true`	The value `7` in row `0`, column `1` is matched with the value `7` in row `1`, column `0`.

`1 2 3 4`
`5 6 7 8`
`5 4 1 2`
	`clearPair(2, 2)`	`1 2 3 4`
		`5 6 7 8`
		`5 4 1 2`
			`false`	There is no element in row `2` or a later row to pair with the value `1`.

`8 1 0 5`
`0 4 3 6`
`3 4 5 8`
	`clearPair(1, 1)`	`8 1 0 5`
		`0 0 3 0`
		`3 4 5 8`
			`true`	The value `4` in row `1`, column `1` is matched with the value `6` in row `1`, column `3`. It could also have been matched with the value `4` in row `2`, column `1`.

`1 7 9`
`2 6 5`
`4 4 4`
	`clearPair(0, 2)`	`0 7 0`
		`2 6 5`
		`4 4 4`
			`true`	The value `9` in row `0`, column `2` is matched with the value `1` in row `0`, column `0`.

Complete method `clearPair`.

```
/**
 * Identifies and clears an element of puzzle that can be paired with
 * the element at the given row and column, as described in part (b)
 * Preconditions: row and col are valid row and column indices in puzzle.
 *  The element at the given row and column is between 1 and 9, inclusive.
 */
public boolean clearPair(int row, int col)
```

STOP

END OF EXAM

4. This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
public class Location
{
   private int theRow;
   private int theCol;

   public Location(int r, int c)
   {
      theRow = r;
      theCol = c;
   }

   public int getRow()
   {  return theRow;  }

   public int getCol()
   {  return theCol;  }
}
```

The following `GridPath` class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the `GridPath` class.

```
public class GridPath
{
    /** Initialized in the constructor with distinct values that never change */
    private int[][] grid;

    /**
     *  Returns the Location representing a neighbor of the grid element at row and col,
     *  as described in part (a)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public Location getNextLoc(int row, int col)
    {  /* to be implemented in part (a) */  }

    /**
     *  Computes and returns the sum of all values on a path through grid, as described in
     *  part (b)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public int sumPath(int row, int col)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered are the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since `3 < 11`
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since `15 < 16`
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below

In the example, the `getNextLoc` method will never be called with row `4` and column `4`, as those values would violate the precondition of the method.

Complete the `getNextLoc` method.

```
/**
 *  Returns the Location representing a neighbor of the grid element at row and col,
 *  as described in part (a)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public Location getNextLoc(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and the last column of `grid` is reached.

For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns `19` because 3 + 2 + 9 + 4 + 0 + 1 = 19.

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

Write the `sumPath` method. Assume `getNextLoc` works as intended, regardless of what you wrote in part (a). You must use `getNextLoc` appropriately in order to receive full credit.

```
/**
 * Computes and returns the sum of all values on a path through grid, as described in
 * part (b)
 * Preconditions: row is a valid row index and col is a valid column index in grid.
 *      row and col do not specify the element in the last row and last column of grid.
 */
public int sumPath(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

STOP

END OF EXAM

4. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
    /** The lights on the board, where true represents on and false represents off.
     */
    private boolean[][] lights;

    /** Constructs a LightBoard object having numRows rows and numCols columns.
     *  Precondition: numRows > 0, numCols > 0
     *  Postcondition: each light has a 40% probability of being set to on.
     */
    public LightBoard(int numRows, int numCols)
    {  /* to be implemented in part (a) */  }

    /** Evaluates a light in row index row and column index col and returns a status
     *  as described in part (b).
     *  Precondition: row and col are valid indexes in lights.
     */
    public boolean evaluateLight(int row, int col)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) Write the constructor for the `LightBoard` class, which initializes `lights` so that each light is set to on with a 40% probability. The notation `lights[r][c]` represents the array element at row `r` and column `c`.

Complete the `LightBoard` constructor below.

```
/** Constructs a LightBoard object having numRows rows and numCols columns.
 *  Precondition: numRows > 0, numCols > 0
 *  Postcondition: each light has a 40% probability of being set to on.
 */
public LightBoard(int numRows, int numCols)
```

(b) Write the method `evaluateLight`, which computes and returns the status of a light at a given row and column based on the following rules.

1. If the light is on, return `false` if the number of lights in its column that are on is even, including the current light.
2. If the light is off, return `true` if the number of lights in its column that are on is divisible by three.
3. Otherwise, return the light's current status.

For example, suppose that `LightBoard sim = new LightBoard(7, 5)` creates a light board with the initial state shown below, where `true` represents a light that is on and `false` represents a light that is off. Lights that are off are shaded.

`lights`

	0	1	2	3	4
0	`true`	`true`	`false`	`true`	`true`
1	`true`	`false`	`false`	`true`	`false`
2	`true`	`false`	`false`	`true`	`true`
3	`true`	`false`	`false`	`false`	`true`
4	`true`	`false`	`false`	`false`	`true`
5	`true`	`true`	`false`	`true`	`true`
6	`false`	`false`	`false`	`false`	`false`

Sample calls to `evaluateLight` are shown below.

Call to `evaluateLight`	**Value Returned**	**Explanation**
`sim.evaluateLight(0, 3);`	`false`	The light is on, and the number of lights that are on in its column is even.
`sim.evaluateLight(6, 0);`	`true`	The light is off, and the number of lights that are on in its column is divisible by 3.
`sim.evaluateLight(4, 1);`	`false`	Returns the light's current status.
`sim.evaluateLight(5, 4);`	`true`	Returns the light's current status.

Class information for this question

```
public class LightBoard
private boolean[][] lights
public LightBoard(int numRows, int numCols)
public boolean evaluateLight(int row, int col)
```

Complete the `evaluateLight` method below.

```
/** Evaluates a light in row index row and column index col and returns a status
 *  as described in part (b).
 *  Precondition: row and col are valid indexes in lights.
 */
public boolean evaluateLight(int row, int col)
```

STOP

END OF EXAM

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

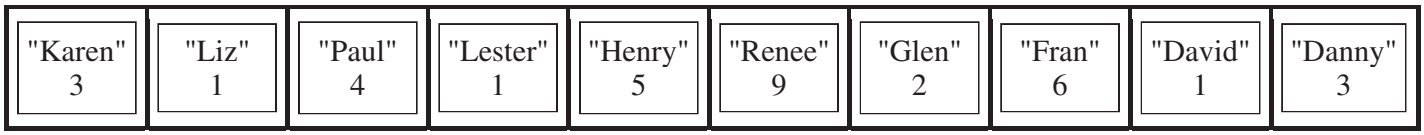
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** 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<Student> roster` contains references to `Student` objects in the following order.



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`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

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)
```

Part (b) begins on page 10.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.12 2D Array Traversals

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS row-major order 行优先 • nested loops 嵌套循环 • array 数组 • 2d array 二维数组 • traverse 遍历

Objective

Build the skills to answer exam questions on **2D array traversals**.

You must be able to:

- **traverse** a 2D array using **nested loops** 嵌套循环
- visit elements in **row-major order** 行优先
- use nested enhanced for loops

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Nested-loop traversal

Use an outer loop over rows and an inner loop over columns:

```
for (int r = 0; r < g.length; r++)
    for (int c = 0; c < g[0].length; c++)
        System.out.print(g[r][c]);
```

■ Row-major order

This visits elements in **row-major order** — it finishes each **row** completely before moving to the next. A 2D array of R rows and C columns has $R \times C$ elements.

2 Practice

2.1 State how many loops are needed to traverse a 2D array. [1]

2.2 State what “row-major order” means. [1]

2.3 State how many elements a 3×4 2D array has. [1]

3 Exam-style questions

3.1 A 2D array is traversed with [1]

- **A** one loop
- **B** nested loops
- **C** no loop
- **D** a switch statement

3.2 Row-major order visits [1]

- **A** each column fully first
- **B** each row fully before the next
- **C** elements at random
- **D** elements backwards

3.3 A 2D array has 3 rows and 5 columns.

(a) State the total number of elements. [1]

(b) State how many loops traverse it. [1]

(c) Name the order that completes each row first. [1]

Solutions

2.1 two (nested loops).

2.2 it visits each row completely before moving to the next row.

2.3 $3 \times 4 = 12$.

3.1 B.

3.2 B.

3.3 (a) 15. (b) two (nested). (c) row-major order.

4. This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
public class Location
{
   private int theRow;
   private int theCol;

   public Location(int r, int c)
   {
      theRow = r;
      theCol = c;
   }

   public int getRow()
   {  return theRow;  }

   public int getCol()
   {  return theCol;  }
}
```

The following `GridPath` class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the `GridPath` class.

```
public class GridPath
{
   /** Initialized in the constructor with distinct values that never change */
   private int[][] grid;

   /**
    *  Returns the Location representing a neighbor of the grid element at row and col,
    *  as described in part (a)
    *  Preconditions: row is a valid row index and col is a valid column index in grid.
    *       row and col do not specify the element in the last row and last column of grid.
    */
   public Location getNextLoc(int row, int col)
   {  /* to be implemented in part (a) */  }

   /**
    *  Computes and returns the sum of all values on a path through grid, as described in
    *  part (b)
    *  Preconditions: row is a valid row index and col is a valid column index in grid.
    *       row and col do not specify the element in the last row and last column of grid.
    */
   public int sumPath(int row, int col)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered are the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since `3 < 11`
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since `15 < 16`
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below

In the example, the `getNextLoc` method will never be called with row `4` and column `4`, as those values would violate the precondition of the method.

Complete the `getNextLoc` method.

```
/**
 *  Returns the Location representing a neighbor of the grid element at row and col,
 *  as described in part (a)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public Location getNextLoc(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and the last column of `grid` is reached.

For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns `19` because 3 + 2 + 9 + 4 + 0 + 1 = 19.

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

Write the `sumPath` method. Assume `getNextLoc` works as intended, regardless of what you wrote in part (a). You must use `getNextLoc` appropriately in order to receive full credit.

```
/**
 *  Computes and returns the sum of all values on a path through grid, as described in
 *  part (b)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public int sumPath(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

STOP

END OF EXAM

4. This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the `Data` class is shown. You will write two methods of the `Data` class.

```
public class Data
{
   public static final int MAX = /* value not shown */;
   private int[][] grid;

   /** Fills all elements of grid with randomly generated values, as described in part (a)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public void repopulate()
   {  /* to be implemented in part (a) */  }

   /** Returns the number of columns in grid that are in increasing order, as described
    *  in part (b)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public int countIncreasingCols()
   {  /* to be implemented in part (b) */  }

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

GO ON TO THE NEXT PAGE.

(a) Write the `repopulate` method, which assigns a newly generated random value to each element of `grid`. Each value is computed to meet all of the following criteria, and all valid values must have an equal chance of being generated.

- The value is between 1 and `MAX`, inclusive.
- The value is divisible by 10.
- The value is not divisible by 100.

Complete the `repopulate` method.

```
/** Fills all elements of grid with randomly generated values, as described in part (a)
 *  Precondition: grid is not null.
 *      grid has at least one element.
 */
public void repopulate()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

(b) Write the `countIncreasingCols` method, which returns the number of columns in `grid` that are in increasing order. A column is considered to be in increasing order if the element in each row after the first row is greater than or equal to the element in the previous row. A column with only one row is considered to be in increasing order.

The following examples show the `countIncreasingCols` return values for possible contents of `grid`.

The return value for the following contents of `grid` is `1`, since the first column is in increasing order but the second and third columns are not.

10	50	40
20	40	20
30	50	30

The return value for the following contents of `grid` is `2`, since the first and third columns are in increasing order but the second and fourth columns are not.

10	540	440	440
220	450	440	190

Complete the `countIncreasingCols` method.

```
/** Returns the number of columns in grid that are in increasing order, as described
 *  in part (b)
 *  Precondition: grid is not null.
 *      grid has at least one element.
 */
public int countIncreasingCols()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Data

public static final int MAX = /* value not shown */
private int[][] grid

public void repopulate()
public int countIncreasingCols()
```

GO ON TO THE NEXT PAGE.

STOP

END OF EXAM

4. This question involves manipulating a two-dimensional array of integers. You will write two `static` methods of the `ArrayResizer` class, which is shown below.

```
public class ArrayResizer
{
    /** Returns true if and only if every value in row r of array2D is non-zero.
     *  Precondition: r is a valid row index in array2D.
     *  Postcondition: array2D is unchanged.
     */
    public static boolean isNonZeroRow(int[][] array2D, int r)
    {  /* to be implemented in part (a) */  }

    /** Returns the number of rows in array2D that contain all non-zero values.
     *  Postcondition: array2D is unchanged.
     */
    public static int numNonZeroRows(int[][] array2D)
    {  /* implementation not shown */  }

    /** Returns a new, possibly smaller, two-dimensional array that contains only rows
     *  from array2D with no zeros, as described in part (b).
     *  Precondition: array2D contains at least one column and at least one row with no zeros.
     *  Postcondition: array2D is unchanged.
     */
    public static int[][] resize(int[][] array2D)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the method `isNonZeroRow`, which returns `true` if and only if all elements in row `r` of a two-dimensional array `array2D` are not equal to zero.

For example, consider the following statement, which initializes a two-dimensional array.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
```

Sample calls to `isNonZeroRow` are shown below.

Call to `isNonZeroRow`	**Value Returned**	**Explanation**
`ArrayResizer.isNonZeroRow(arr, 0)`	`false`	At least one value in row `0` is zero.
`ArrayResizer.isNonZeroRow(arr, 1)`	`true`	All values in row `1` are non-zero.
`ArrayResizer.isNonZeroRow(arr, 2)`	`false`	At least one value in row `2` is zero.
`ArrayResizer.isNonZeroRow(arr, 3)`	`true`	All values in row `3` are non-zero.

Complete the `isNonZeroRow` method.

```
/** Returns true if and only if every value in row r of array2D is non-zero.
 *  Precondition: r is a valid row index in array2D.
 *  Postcondition: array2D is unchanged.
 */
public static boolean isNonZeroRow(int[][] array2D, int r)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the method `resize`, which returns a new two-dimensional array containing only rows from `array2D` with all non-zero values. The elements in the new array should appear in the same order as the order in which they appeared in the original array.

The following code segment initializes a two-dimensional array and calls the `resize` method.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
int[][] smaller = ArrayResizer.resize(arr);
```

When the code segment completes, the following will be the contents of `smaller`.

```
{{1, 3, 2}, {4, 5, 6}}
```

A helper method, `numNonZeroRows`, has been provided for you. The method returns the number of rows in its two-dimensional array parameter that contain no zero values.

Complete the `resize` method. Assume that `isNonZeroRow` works as specified, regardless of what you wrote in part (a). You must use `numNonZeroRows` and `isNonZeroRow` appropriately to receive full credit.

```
/** Returns a new, possibly smaller, two-dimensional array that contains only rows from array2D
 *  with no zeros, as described in part (b).
 *  Precondition: array2D contains at least one column and at least one row with no zeros.
 *  Postcondition: array2D is unchanged.
 */
public static int[][] resize(int[][] array2D)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class ArrayResizer

public static boolean isNonZeroRow(int[][] array2D, int r)
public static int numNonZeroRows(int[][] array2D)
public static int[][] resize(int[][] array2D)
```

STOP

END OF EXAM

4. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
    /** The lights on the board, where true represents on and false represents off.
     */
    private boolean[][] lights;

    /** Constructs a LightBoard object having numRows rows and numCols columns.
     *  Precondition: numRows > 0, numCols > 0
     *  Postcondition: each light has a 40% probability of being set to on.
     */
    public LightBoard(int numRows, int numCols)
    {  /* to be implemented in part (a) */  }

    /** Evaluates a light in row index row and column index col and returns a status
     *  as described in part (b).
     *  Precondition: row and col are valid indexes in lights.
     */
    public boolean evaluateLight(int row, int col)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) Write the constructor for the `LightBoard` class, which initializes `lights` so that each light is set to on with a 40% probability. The notation `lights[r][c]` represents the array element at row `r` and column `c`.

Complete the `LightBoard` constructor below.

```
/** Constructs a LightBoard object having numRows rows and numCols columns.
 *  Precondition: numRows > 0, numCols > 0
 *  Postcondition: each light has a 40% probability of being set to on.
 */
public LightBoard(int numRows, int numCols)
```

(b) Write the method `evaluateLight`, which computes and returns the status of a light at a given row and column based on the following rules.

1. If the light is on, return `false` if the number of lights in its column that are on is even, including the current light.
2. If the light is off, return `true` if the number of lights in its column that are on is divisible by three.
3. Otherwise, return the light's current status.

For example, suppose that `LightBoard sim = new LightBoard(7, 5)` creates a light board with the initial state shown below, where `true` represents a light that is on and `false` represents a light that is off. Lights that are off are shaded.

`lights`

	0	1	2	3	4
0	`true`	`true`	`false`	`true`	`true`
1	`true`	`false`	`false`	`true`	`false`
2	`true`	`false`	`false`	`true`	`true`
3	`true`	`false`	`false`	`false`	`true`
4	`true`	`false`	`false`	`false`	`true`
5	`true`	`true`	`false`	`true`	`true`
6	`false`	`false`	`false`	`false`	`false`

Sample calls to `evaluateLight` are shown below.

Call to `evaluateLight`	**Value Returned**	**Explanation**
`sim.evaluateLight(0, 3);`	`false`	The light is on, and the number of lights that are on in its column is even.
`sim.evaluateLight(6, 0);`	`true`	The light is off, and the number of lights that are on in its column is divisible by 3.
`sim.evaluateLight(4, 1);`	`false`	Returns the light's current status.
`sim.evaluateLight(5, 4);`	`true`	Returns the light's current status.

Class information for this question

```
public class LightBoard
private boolean[][] lights
public LightBoard(int numRows, int numCols)
public boolean evaluateLight(int row, int col)
```

Complete the `evaluateLight` method below.

```
/** Evaluates a light in row index row and column index col and returns a status
 *  as described in part (b).
 *  Precondition: row and col are valid indexes in lights.
 */
public boolean evaluateLight(int row, int col)
```

STOP

END OF EXAM

4. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {  /* implementation not shown */  }


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {  /* to be implemented in part (b) */  }
```

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row `r` and column `c`.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable `result` will have the following contents.

```
result: {1, 4, 7, 5}
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D
```

(b) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a *Latin square*, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The `ArrayTester` class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array `arr` contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in `arr1` appears in `arr2`. You do not need to write the code for these methods.

Class information for this question

```
public class ArrayTester

public static int[] getColumn(int[][] arr2D, int c)
public static boolean hasAllValues(int[] arr1, int[] arr2)
public static boolean containsDuplicates(int[] arr)
public static boolean isLatin(int[][] square)
```

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

STOP

END OF EXAM

4. This question involves reasoning about a two-dimensional (2D) array of integers. You will write two static methods, both of which are in a single enclosing class named `Successors` (not shown). These methods process a 2D integer array that contains consecutive values. Each of these integers may be in any position in the 2D integer array. For example, the following 2D integer array with 3 rows and 4 columns contains the integers 5 through 16, inclusive.

2D Integer Array

	0	1	2	3
0	15	5	9	10
1	12	16	11	6
2	14	8	13	7

The following `Position` class is used to represent positions in the integer array. The notation `(r,c)` will be used to refer to a `Position` object with row `r` and column `c`.

```
public class Position
{
    /** Constructs a Position object with row r and column c. */
    public Position(int r, int c)
    {  /* implementation not shown */  }

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

(a) Write a `static` method `findPosition` that takes an integer value and a 2D integer array and returns the position of the integer in the given 2D integer array. If the integer is not an element of the 2D integer array, the method returns `null`.

For example, assume that array `arr` is the 2D integer array shown at the beginning of the question.

- The call `findPosition(8, arr)` would return the `Position` object `(2,1)` because the value `8` appears in `arr` at row 2 and column 1.
- The call `findPosition(17, arr)` would return `null` because the value `17` does not appear in `arr`.

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
```

Part (b) begins on page 18.

(b) Write a `static` method `getSuccessorArray` that returns a 2D successor array of positions created from a given 2D integer array.

The *successor* of an integer value is the integer that is one greater than that value. For example, the successor of 8 is 9. A 2D *successor array* shows the position of the successor of each element in a given 2D integer array. The 2D successor array has the same dimensions as the given 2D integer array. Each element in the 2D successor array is the position (row, column) of the corresponding 2D integer array element's successor. The largest element in the 2D integer array does not have a successor in the 2D integer array, so its corresponding position in the 2D successor array is `null`.

The following diagram shows a 2D integer array and its corresponding 2D successor array. To illustrate the successor relationship, the values 8 and 9 in the 2D integer array are shaded. In the 2D successor array, the shaded element shows that the position of the successor of 8 is `(0,2)` in the 2D integer array. The largest value in the 2D integer array is 16, so its corresponding element in the 2D successor array is `null`.

2D Integer Array

	0	1	2	3
0	15	5	9	10
1	12	16	11	6
2	14	8	13	7

2D Successor Array

	0	1	2	3
0	(1,1)	(1,3)	(0,3)	(1,2)
1	(2,2)	null	(1,0)	(2,3)
2	(0,0)	(0,2)	(2,0)	(2,1)

Class information for this question

```
public class Position

public Position(int r, int c)

public class Successors

public static Position findPosition(int num, int[][] intArr)
public static Position[][] getSuccessorArray(int[][] intArr)
```

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
```

STOP

END OF EXAM

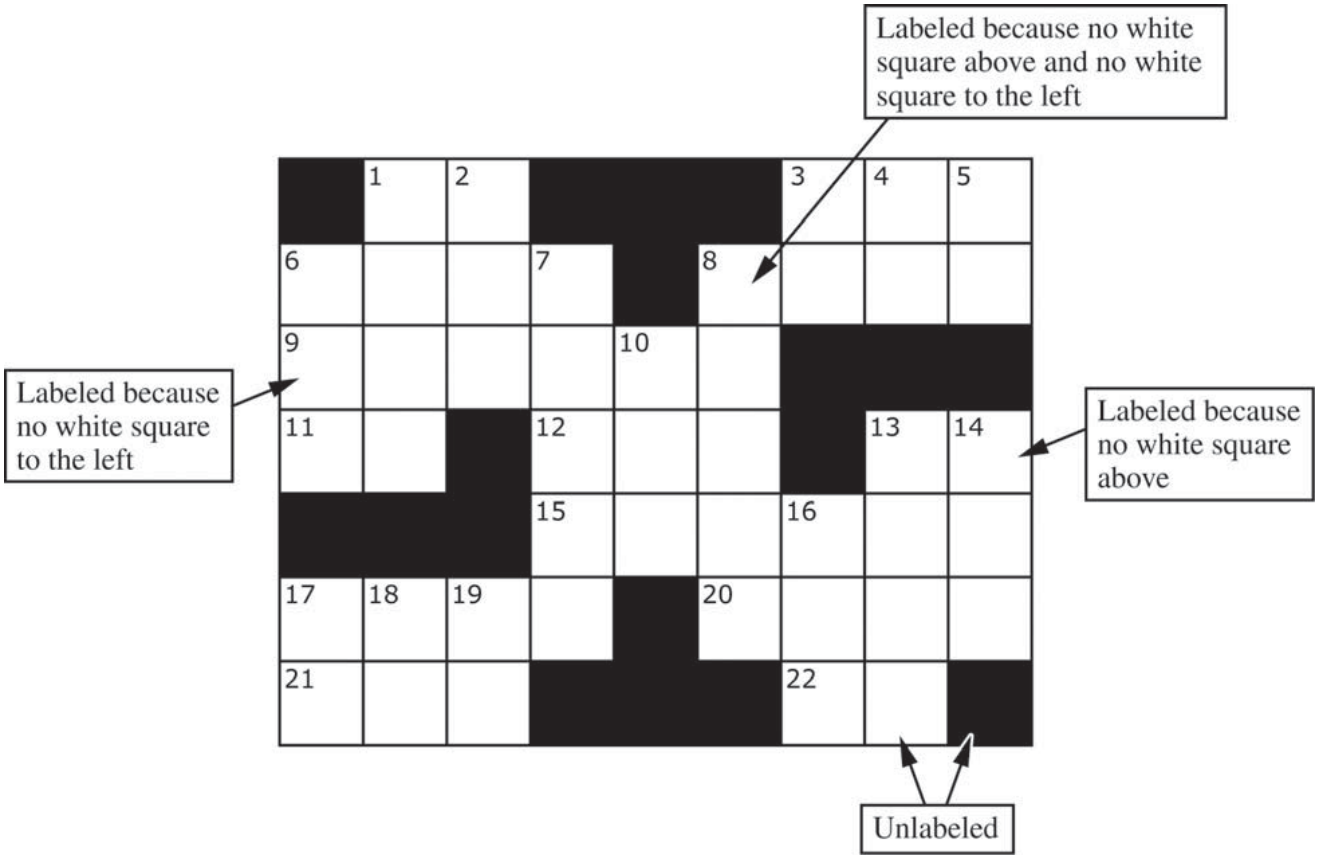
3. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

   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.



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.

```
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.
}
```

Part (a) begins on page 14.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (b) begins on page 16.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the Java 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 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`.

	`arr1`				Value returned by `arraySum(arr1)`
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (c) begins on page 6.

(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)
```

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

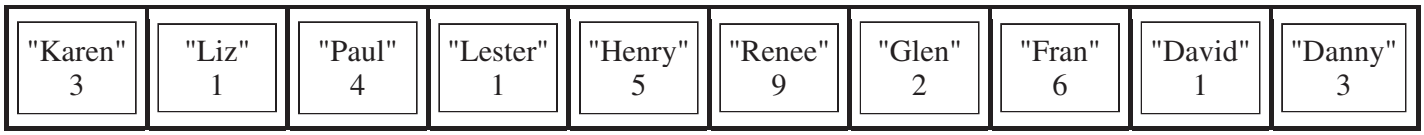
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** 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<Student> roster` contains references to `Student` objects in the following order.



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`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

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)
```

Part (b) begins on page 10.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.13 Implementing 2D Array Algorithms

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS 2d array 二维数组 • array 数组

Objective

Build the skills to answer exam questions on **implementing 2D array algorithms**.

You must be able to:

- sum, count, or search the elements of a **2D array** 二维数组
- find the maximum or minimum across a whole 2D array
- process a single **row** or a single **column**

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Summing a 2D array

Use nested loops and accumulate a total:

```
int sum = 0;
for (int r = 0; r < g.length; r++)
    for (int c = 0; c < g[0].length; c++)
        sum += g[r][c];
```

■ One row or column

To process a single **row** `r`, loop over `g[r]`. To process a single **column** `c`, fix `c` and loop the **row** index.

2 Practice

2.1 Describe how to sum every element of a 2D array. [1]

2.2 State how to process a single row `r` of a 2D array `g`. [1]

2.3 For `int[][] g = {{1, 2}, {3, 4}};`, find the sum of all elements. [2]

3 Exam-style questions

3.1 To find the maximum across a whole 2D array, you [1]

- **A** sum all the elements
- **B** track the largest value over all elements
- **C** sort the rows
- **D** count the elements

3.2 To sum a single column, you fix the [1]

- **A** row index
- **B** column index
- **C** both indices
- **D** neither index

3.3 `int[][] g = {{2, 4}, {6, 8}};`.

(a) State the sum of all elements. [1]

(b) State the sum of row 0. [1]

(c) State the sum of column 1. [1]

Solutions

2.1 use nested loops to add every element to a running total.

2.2 loop over the elements of `g[r]`.

2.3 $1 + 2 + 3 + 4 = 10$.

3.1 B.

3.2 B.

3.3 (a) 20. (b) $2 + 4 = 6$. (c) $4 + 8 = 12$.

4. The `Space` class is used to represent the spaces on a game board. A partial definition of the `Space` class is shown.

```
public class Space
{
   /**
    *  Returns the color of the space
    */
   public String getColor()
   { /* implementation not shown */ }

   /**
    *  Returns the point value of the space
    */
   public int getPoints()
   { /* implementation not shown */ }

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

The `GameBoard` class maintains a two-dimensional array of `Space` objects that represents a game board. A partial definition of the `GameBoard` class is shown.

```
public class GameBoard
{
   private Space[][] board;

   /**
    *  Returns the point value of the row in board specified by the
    *  parameter. The point value is the sum of the points in the row,
    *  or two times the sum of the points in the row if all spaces in
    *  the row are the same color.
    *  Preconditions: No elements of board are null.
    *                 board has at least two rows and
    *                 at least two columns.
    *                 targetRow is a valid row index.
    */
   public int getPointsForRow(int targetRow)
   { /* to be implemented */ }

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

When an element of the two-dimensional array `board` is accessed, the first index is used to specify the row and the second index is used to specify the column.

Write the `GameBoard` method `getPointsForRow`. The method should return the point value of the spaces in the row specified by the parameter `targetRow`. The point value of a row is determined as follows.

- If not all spaces in the row are the same color, the point value for the row is the sum of the points in the row.
- If all spaces in the row are the same color, the point value for the row is two times the sum of the points in the row.

For example, suppose `board` has the following contents. For each element, the first value is the color and the second value is the point value.

	0	1	2	3	4
0	"orange" 100	"red" 100	"blue" 500	"green" 500	"red" 100
1	"red" 300	"blue" 200	"blue" 400	"red" 100	"green" 100
2	"red" 200	"red" 300	"red" 100	"red" 200	"red" 200
3	"green" 100	"blue" 100	"blue" 200	"blue" 100	"blue" 200

For these contents of `board`, the expected behavior of `getPointsForRow` is as follows.

- The call `getPointsForRow(0)` should return `1300`. Not all spaces in this row are the same color, so the sum of the points in the row is returned.
- The call `getPointsForRow(2)` should return `2000`. The sum of the points in row 2 is `1000`, and all spaces in this row are the same color, so two times this sum is returned.

Complete method `getPointsForRow`.

```
/**
 *  Returns the point value of the row in board specified by the
 *  parameter. The point value is the sum of the points in the row,
 *  or two times the sum of the points in the row if all spaces in
 *  the row are the same color.
 *  Preconditions: No elements of board are null.
 *                 board has at least two rows and
 *                 at least two columns.
 *                 targetRow is a valid row index.
 */
public int getPointsForRow(int targetRow)
```

STOP

END OF EXAM

4. This question involves reasoning about a number puzzle that is represented as a two-dimensional array of integers. Each element of the array initially contains a value between 1 and 9, inclusive. Solving the puzzle involves clearing pairs of array elements by setting them to 0. Two elements can be cleared if their values sum to 10 or if they have the same value. The puzzle is considered solved if all elements of the array are cleared.

You will write the constructor and one method of the `SumOrSameGame` class, which contains the methods that manipulate elements of the puzzle.

```
public class SumOrSameGame
{
  private int[][] puzzle;

  /**
   * Creates a two-dimensional array and fills it with random integers,
   * as described in part (a)
   * Precondition: numRows > 0; numCols > 0
   */
  public SumOrSameGame(int numRows, int numCols)
  { /* to be implemented in part (a) */ }

  /**
   * Identifies and clears an element of puzzle that can be paired with
   * the element at the given row and column, as described in part (b)
   * Preconditions: row and col are valid row and column indices in puzzle.
   *  The element at the given row and column is between 1 and 9, inclusive.
   */
  public boolean clearPair(int row, int col)
  { /* to be implemented in part (b) */ }

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

A. Write the constructor for the `SumOrSameGame` class. The constructor initializes the instance variable `puzzle` to be a two-dimensional integer array with the number of rows and columns specified by the parameters `numRows` and `numCols`, respectively. Array elements are initialized with random integers between `1` and `9`, inclusive, each with an equal chance of being assigned to each element of `puzzle`.

When an element of the two-dimensional array is accessed, the first index is used to specify the row and the second index is used to specify the column.

Complete the `SumOrSameGame` constructor.

```
/**
 * Creates a two-dimensional array and fills it with random integers,
 * as described in part (a)
 * Precondition: numRows > 0; numCols > 0
 */
public SumOrSameGame(int numRows, int numCols)
```

B. Write the `clearPair` method, which takes a valid row index and valid column index as its parameters. The array element specified by those indices, which has a value between `1` and `9`, inclusive, is compared to other array elements in `puzzle` in an attempt to pair it with another array element that meets both of the following conditions.

- The row index of the second element is greater than or equal to the parameter `row`.
- The two elements have equal values or have values that sum to 10.

If such an array element is found, both array elements of the pair are cleared (set to `0`) and the method returns `true`. If more than one such array element is found, any one of those identified array elements can be used to complete the pair and can be cleared. If no such array element is found, no changes are made to `puzzle` and the method returns `false`.

The following table shows the possible results of several calls to `clearPair`.

`puzzle` before call to `clearPair`	Method Call	`puzzle` after call to `clearPair`	Return Value	Explanation

`0 7 9 0`
`7 4 1 6`
`8 4 1 8`
	`clearPair(0, 1)`	`0 0 9 0`
		`0 4 1 6`
		`8 4 1 8`
			`true`	The value `7` in row `0`, column `1` is matched with the value `7` in row `1`, column `0`.

`1 2 3 4`
`5 6 7 8`
`5 4 1 2`
	`clearPair(2, 2)`	`1 2 3 4`
		`5 6 7 8`
		`5 4 1 2`
			`false`	There is no element in row `2` or a later row to pair with the value `1`.

`8 1 0 5`
`0 4 3 6`
`3 4 5 8`
	`clearPair(1, 1)`	`8 1 0 5`
		`0 0 3 0`
		`3 4 5 8`
			`true`	The value `4` in row `1`, column `1` is matched with the value `6` in row `1`, column `3`. It could also have been matched with the value `4` in row `2`, column `1`.

`1 7 9`
`2 6 5`
`4 4 4`
	`clearPair(0, 2)`	`0 7 0`
		`2 6 5`
		`4 4 4`
			`true`	The value `9` in row `0`, column `2` is matched with the value `1` in row `0`, column `0`.

Complete method `clearPair`.

```
/**
 * Identifies and clears an element of puzzle that can be paired with
 * the element at the given row and column, as described in part (b)
 * Preconditions: row and col are valid row and column indices in puzzle.
 *  The element at the given row and column is between 1 and 9, inclusive.
 */
public boolean clearPair(int row, int col)
```

STOP

END OF EXAM

4. This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
public class Location
{
   private int theRow;
   private int theCol;

   public Location(int r, int c)
   {
      theRow = r;
      theCol = c;
   }

   public int getRow()
   {  return theRow;  }

   public int getCol()
   {  return theCol;  }
}
```

The following `GridPath` class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the `GridPath` class.

```
public class GridPath
{
    /** Initialized in the constructor with distinct values that never change */
    private int[][] grid;

    /**
     *  Returns the Location representing a neighbor of the grid element at row and col,
     *  as described in part (a)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public Location getNextLoc(int row, int col)
    {  /* to be implemented in part (a) */  }

    /**
     *  Computes and returns the sum of all values on a path through grid, as described in
     *  part (b)
     *  Preconditions: row is a valid row index and col is a valid column index in grid.
     *       row and col do not specify the element in the last row and last column of grid.
     */
    public int sumPath(int row, int col)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered are the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since `3 < 11`
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since `15 < 16`
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below

In the example, the `getNextLoc` method will never be called with row `4` and column `4`, as those values would violate the precondition of the method.

Complete the `getNextLoc` method.

```
/**
 *  Returns the Location representing a neighbor of the grid element at row and col,
 *  as described in part (a)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public Location getNextLoc(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and the last column of `grid` is reached.

For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns `19` because 3 + 2 + 9 + 4 + 0 + 1 = 19.

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

Write the `sumPath` method. Assume `getNextLoc` works as intended, regardless of what you wrote in part (a). You must use `getNextLoc` appropriately in order to receive full credit.

```
/**
 *  Computes and returns the sum of all values on a path through grid, as described in
 *  part (b)
 *  Preconditions: row is a valid row index and col is a valid column index in grid.
 *       row and col do not specify the element in the last row and last column of grid.
 */
public int sumPath(int row, int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Location

private int theRow
private int theCol

public Location(int r, int c)
public int getRow()
public int getCol()

public class GridPath

private int[][] grid

public Location getNextLoc(int row, int col)
public int sumPath(int row, int col)
```

STOP

END OF EXAM

4. This question involves pieces of candy in a box. The `Candy` class represents a single piece of candy.

```
public class Candy
{
    /** Returns a String representing the flavor of this piece of candy */
    public String getFlavor()
    {  /* implementation not shown */  }

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

The `BoxOfCandy` class represents a candy box where the candy is arranged in a rectangular grid. The instance variable of the class, `box`, is a rectangular two-dimensional array of `Candy` objects. A location in the candy box may contain a piece of candy or may be empty. A piece of candy is represented by a `Candy` object. An empty location is represented by `null`.

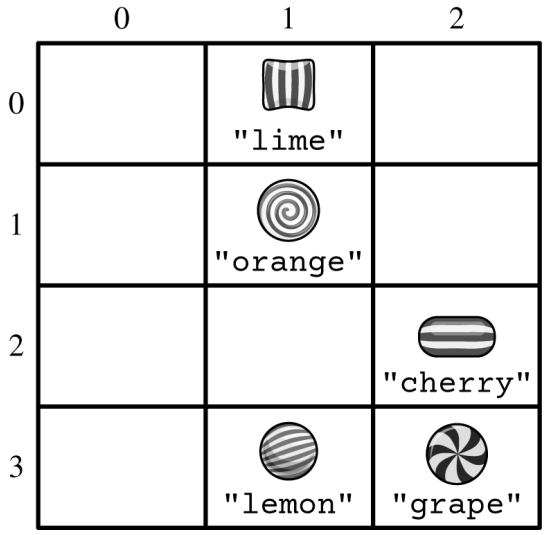
You will write two methods of the `BoxOfCandy` class.

```
public class BoxOfCandy
{
    /** box contains at least one row and is initialized in the constructor. */
    private Candy[][] box;

    /**
     *  Moves one piece of candy in column col, if necessary and possible, so that the box
     *  element in row 0 of column col contains a piece of candy, as described in part (a).
     *  Returns false if there is no piece of candy in column col and returns true otherwise.
     *  Precondition: col is a valid column index in box.
     */
    public boolean moveCandyToFirstRow(int col)
    {  /* to be implemented in part (a) */  }

    /**
     *  Removes from box and returns a piece of candy with flavor specified by the parameter, or
     *  returns null if no such piece is found, as described in part (b)
     */
    public Candy removeNextByFlavor(String flavor)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `moveCandyToFirstRow` method, which attempts to ensure that the `box` element at row `0` and column `col` contains a piece of candy, using the following steps.

- If the element at row `0` and column `col` already contains a piece of candy, then `box` is unchanged and the method returns `true`.
- If the element at row `0` and column `col` does not contain a piece of candy, then the method searches the remaining rows of column `col` for a piece of candy. If a piece of candy can be found in column `col`, it is moved to row `0`, its previous location is set to `null`, and the method returns `true`; otherwise, the method returns `false`.

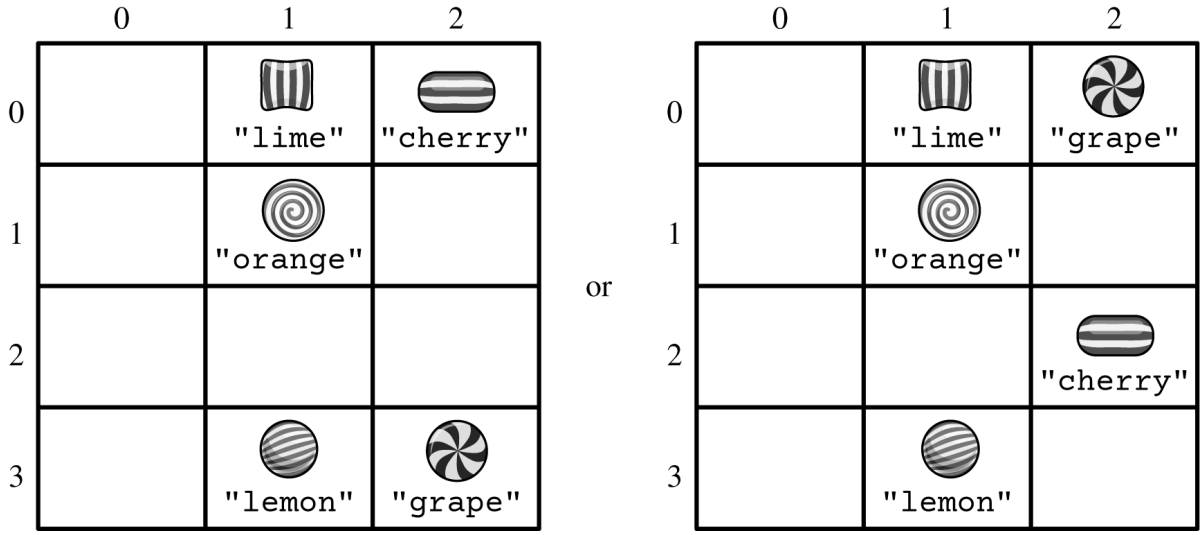
In the following example, the grid represents the contents of `box`. An empty square in the grid is `null` in `box`. A non-empty square in the grid represents a `box` element that contains a `Candy` object. The string in the square of the grid indicates the flavor of the piece of candy.



The method call `moveCandyToFirstRow(0)` returns `false` because the `box` element at row `0` and column `0` does not contain a piece of candy and there are no pieces of candy in column `0` that can be moved to row `0`. The contents of `box` are unchanged.

The method call `moveCandyToFirstRow(1)` returns `true` because the `box` element at row `0` and column `1` already contains a piece of candy. The contents of `box` are unchanged.

The method call `moveCandyToFirstRow(2)` moves one of the two pieces of candy in column `2` to row `0` of column `2`, sets the previous location of the piece of candy that was moved to `null`, and returns `true`. The new contents of `box` could be either of the following.



Complete the `moveCandyToFirstRow` method.

```
/**
 * Moves one piece of candy in column col, if necessary and possible, so that the box
 * element in row 0 of column col contains a piece of candy, as described in part (a).
 * Returns false if there is no piece of candy in column col and returns true otherwise.
 * Precondition: col is a valid column index in box.
 */
public boolean moveCandyToFirstRow(int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Candy
public String getFlavor()
public class BoxOfCandy
private Candy[][] box
public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

(b) Write the `removeNextByFlavor` method, which attempts to remove and return one piece of candy from the box. The piece of candy to be removed is the first piece of candy with a flavor equal to the parameter `flavor` that is encountered while traversing the candy box in the following order: the last row of the box is traversed from left to right, then the next-to-last row of the box is traversed from left to right, etc., until either a piece of candy with the desired flavor is found or until the entire candy box has been searched.

If the `removeNextByFlavor` method finds a `Candy` object with the desired flavor, the corresponding `box` element is assigned `null`, all other `box` elements are unchanged, and the removed `Candy` object is returned. Otherwise, `box` is unchanged and the method returns `null`.

The following examples show three consecutive calls to the `removeNextByFlavor` method. The traversal of the candy box always begins in the last row and first column of the box.

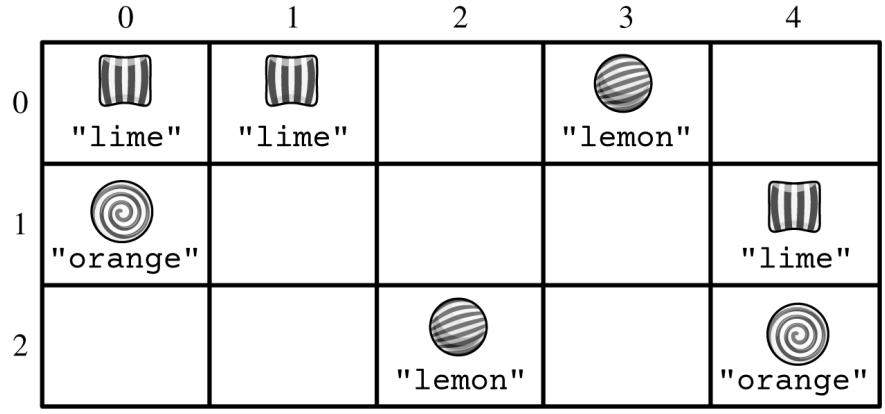
The following grid shows the contents of `box` before any of the `removeNextByFlavor` method calls.

	0	1	2	3	4
0	 "lime"	 "lime"		 "lemon"	
1	 "orange"			 "lime"	 "lime"
2	 "cherry"		 "lemon"		 "orange"

The method call `removeNextByFlavor("cherry")` removes and returns the `Candy` object located in row `2` and column `0`. The following grid shows the updated contents of `box`.

	0	1	2	3	4
0	 "lime"	 "lime"		 "lemon"	
1	 "orange"			 "lime"	 "lime"
2			 "lemon"		 "orange"

The method call `removeNextByFlavor("lime")` removes and returns the `Candy` object located in row `1` and column `3`. The following grid shows the updated contents of `box`.



The method call `removeNextByFlavor("grape")` returns `null` because no grape-flavored candy is found. The contents of `box` are unchanged.

Complete the `removeNextByFlavor` method.

```
/**
 *  Removes from box and returns a piece of candy with flavor specified by the parameter, or
 *  returns null if no such piece is found, as described in part (b)
 */
public Candy removeNextByFlavor(String flavor)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Candy

public String getFlavor()

public class BoxOfCandy

private Candy[][] box

public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

STOP

END OF EXAM

4. This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the `Data` class is shown. You will write two methods of the `Data` class.

```
public class Data
{
   public static final int MAX = /* value not shown */;
   private int[][] grid;

   /** Fills all elements of grid with randomly generated values, as described in part (a)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public void repopulate()
   {  /* to be implemented in part (a) */  }

   /** Returns the number of columns in grid that are in increasing order, as described
    *  in part (b)
    *  Precondition: grid is not null.
    *       grid has at least one element.
    */
   public int countIncreasingCols()
   {  /* to be implemented in part (b) */  }

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

GO ON TO THE NEXT PAGE.

(a) Write the `repopulate` method, which assigns a newly generated random value to each element of `grid`. Each value is computed to meet all of the following criteria, and all valid values must have an equal chance of being generated.

- The value is between 1 and `MAX`, inclusive.
- The value is divisible by 10.
- The value is not divisible by 100.

Complete the `repopulate` method.

```
/** Fills all elements of grid with randomly generated values, as described in part (a)
 *  Precondition: grid is not null.
 *      grid has at least one element.
 */
public void repopulate()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

(b) Write the `countIncreasingCols` method, which returns the number of columns in `grid` that are in increasing order. A column is considered to be in increasing order if the element in each row after the first row is greater than or equal to the element in the previous row. A column with only one row is considered to be in increasing order.

The following examples show the `countIncreasingCols` return values for possible contents of `grid`.

The return value for the following contents of `grid` is `1`, since the first column is in increasing order but the second and third columns are not.

10	50	40
20	40	20
30	50	30

The return value for the following contents of `grid` is `2`, since the first and third columns are in increasing order but the second and fourth columns are not.

10	540	440	440
220	450	440	190

Complete the `countIncreasingCols` method.

```
/** Returns the number of columns in grid that are in increasing order, as described
 *  in part (b)
 *  Precondition: grid is not null.
 *       grid has at least one element.
 */
public int countIncreasingCols()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Data

public static final int MAX = /* value not shown */
private int[][] grid

public void repopulate()
public int countIncreasingCols()
```

GO ON TO THE NEXT PAGE.

STOP

END OF EXAM

4. This question involves manipulating a two-dimensional array of integers. You will write two `static` methods of the `ArrayResizer` class, which is shown below.

```
public class ArrayResizer
{
    /** Returns true if and only if every value in row r of array2D is non-zero.
     *  Precondition: r is a valid row index in array2D.
     *  Postcondition: array2D is unchanged.
     */
    public static boolean isNonZeroRow(int[][] array2D, int r)
    {  /* to be implemented in part (a) */  }

    /** Returns the number of rows in array2D that contain all non-zero values.
     *  Postcondition: array2D is unchanged.
     */
    public static int numNonZeroRows(int[][] array2D)
    {  /* implementation not shown */  }

    /** Returns a new, possibly smaller, two-dimensional array that contains only rows
     *  from array2D with no zeros, as described in part (b).
     *  Precondition: array2D contains at least one column and at least one row with no zeros.
     *  Postcondition: array2D is unchanged.
     */
    public static int[][] resize(int[][] array2D)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the method `isNonZeroRow`, which returns `true` if and only if all elements in row `r` of a two-dimensional array `array2D` are not equal to zero.

For example, consider the following statement, which initializes a two-dimensional array.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
```

Sample calls to `isNonZeroRow` are shown below.

Call to `isNonZeroRow`	**Value Returned**	**Explanation**
`ArrayResizer.isNonZeroRow(arr, 0)`	`false`	At least one value in row `0` is zero.
`ArrayResizer.isNonZeroRow(arr, 1)`	`true`	All values in row `1` are non-zero.
`ArrayResizer.isNonZeroRow(arr, 2)`	`false`	At least one value in row `2` is zero.
`ArrayResizer.isNonZeroRow(arr, 3)`	`true`	All values in row `3` are non-zero.

Complete the `isNonZeroRow` method.

```
/** Returns true if and only if every value in row r of array2D is non-zero.
 *  Precondition: r is a valid row index in array2D.
 *  Postcondition: array2D is unchanged.
 */
public static boolean isNonZeroRow(int[][] array2D, int r)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the method `resize`, which returns a new two-dimensional array containing only rows from `array2D` with all non-zero values. The elements in the new array should appear in the same order as the order in which they appeared in the original array.

The following code segment initializes a two-dimensional array and calls the `resize` method.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
int[][] smaller = ArrayResizer.resize(arr);
```

When the code segment completes, the following will be the contents of `smaller`.

```
{{1, 3, 2}, {4, 5, 6}}
```

A helper method, `numNonZeroRows`, has been provided for you. The method returns the number of rows in its two-dimensional array parameter that contain no zero values.

Complete the `resize` method. Assume that `isNonZeroRow` works as specified, regardless of what you wrote in part (a). You must use `numNonZeroRows` and `isNonZeroRow` appropriately to receive full credit.

```
/** Returns a new, possibly smaller, two-dimensional array that contains only rows from array2D
 *  with no zeros, as described in part (b).
 *  Precondition: array2D contains at least one column and at least one row with no zeros.
 *  Postcondition: array2D is unchanged.
 */
public static int[][] resize(int[][] array2D)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class ArrayResizer

public static boolean isNonZeroRow(int[][] array2D, int r)
public static int numNonZeroRows(int[][] array2D)
public static int[][] resize(int[][] array2D)
```

STOP

END OF EXAM

4. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
    /** The lights on the board, where true represents on and false represents off.
     */
    private boolean[][] lights;

    /** Constructs a LightBoard object having numRows rows and numCols columns.
     *  Precondition: numRows > 0, numCols > 0
     *  Postcondition: each light has a 40% probability of being set to on.
     */
    public LightBoard(int numRows, int numCols)
    {  /* to be implemented in part (a) */  }

    /** Evaluates a light in row index row and column index col and returns a status
     *  as described in part (b).
     *  Precondition: row and col are valid indexes in lights.
     */
    public boolean evaluateLight(int row, int col)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) Write the constructor for the `LightBoard` class, which initializes `lights` so that each light is set to on with a 40% probability. The notation `lights[r][c]` represents the array element at row `r` and column `c`.

Complete the `LightBoard` constructor below.

```
/** Constructs a LightBoard object having numRows rows and numCols columns.
 *  Precondition: numRows > 0, numCols > 0
 *  Postcondition: each light has a 40% probability of being set to on.
 */
public LightBoard(int numRows, int numCols)
```

(b) Write the method `evaluateLight`, which computes and returns the status of a light at a given row and column based on the following rules.

1. If the light is on, return `false` if the number of lights in its column that are on is even, including the current light.
2. If the light is off, return `true` if the number of lights in its column that are on is divisible by three.
3. Otherwise, return the light's current status.

For example, suppose that `LightBoard sim = new LightBoard(7, 5)` creates a light board with the initial state shown below, where `true` represents a light that is on and `false` represents a light that is off. Lights that are off are shaded.

`lights`

	0	1	2	3	4
0	`true`	`true`	`false`	`true`	`true`
1	`true`	`false`	`false`	`true`	`false`
2	`true`	`false`	`false`	`true`	`true`
3	`true`	`false`	`false`	`false`	`true`
4	`true`	`false`	`false`	`false`	`true`
5	`true`	`true`	`false`	`true`	`true`
6	`false`	`false`	`false`	`false`	`false`

Sample calls to `evaluateLight` are shown below.

Call to `evaluateLight`	**Value Returned**	**Explanation**
`sim.evaluateLight(0, 3);`	`false`	The light is on, and the number of lights that are on in its column is even.
`sim.evaluateLight(6, 0);`	`true`	The light is off, and the number of lights that are on in its column is divisible by 3.
`sim.evaluateLight(4, 1);`	`false`	Returns the light's current status.
`sim.evaluateLight(5, 4);`	`true`	Returns the light's current status.

Class information for this question

```
public class LightBoard
private boolean[][] lights
public LightBoard(int numRows, int numCols)
public boolean evaluateLight(int row, int col)
```

Complete the `evaluateLight` method below.

```
/** Evaluates a light in row index row and column index col and returns a status
 *  as described in part (b).
 *  Precondition: row and col are valid indexes in lights.
 */
public boolean evaluateLight(int row, int col)
```

STOP

END OF EXAM

4. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {  /* implementation not shown */  }


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {  /* to be implemented in part (b) */  }
```

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row `r` and column `c`.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable `result` will have the following contents.

```
result: {1, 4, 7, 5}
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D
```

(b) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a *Latin square*, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The `ArrayTester` class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array `arr` contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in `arr1` appears in `arr2`. You do not need to write the code for these methods.

Class information for this question

```
public class ArrayTester

public static int[] getColumn(int[][] arr2D, int c)
public static boolean hasAllValues(int[] arr1, int[] arr2)
public static boolean containsDuplicates(int[] arr)
public static boolean isLatin(int[][] square)
```

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

STOP

END OF EXAM

4. This question involves reasoning about a two-dimensional (2D) array of integers. You will write two static methods, both of which are in a single enclosing class named `Successors` (not shown). These methods process a 2D integer array that contains consecutive values. Each of these integers may be in any position in the 2D integer array. For example, the following 2D integer array with 3 rows and 4 columns contains the integers 5 through 16, inclusive.

2D Integer Array

	0	1	2	3
0	15	5	9	10
1	12	16	11	6
2	14	8	13	7

The following `Position` class is used to represent positions in the integer array. The notation `(r,c)` will be used to refer to a `Position` object with row `r` and column `c`.

```
public class Position
{
    /** Constructs a Position object with row r and column c. */
    public Position(int r, int c)
    {  /* implementation not shown */  }

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

(a) Write a `static` method `findPosition` that takes an integer value and a 2D integer array and returns the position of the integer in the given 2D integer array. If the integer is not an element of the 2D integer array, the method returns `null`.

For example, assume that array `arr` is the 2D integer array shown at the beginning of the question.

- The call `findPosition(8, arr)` would return the `Position` object `(2,1)` because the value `8` appears in `arr` at row 2 and column 1.
- The call `findPosition(17, arr)` would return `null` because the value `17` does not appear in `arr`.

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
```

Part (b) begins on page 18.

(b) Write a `static` method `getSuccessorArray` that returns a 2D successor array of positions created from a given 2D integer array.

The *successor* of an integer value is the integer that is one greater than that value. For example, the successor of 8 is 9. A 2D *successor array* shows the position of the successor of each element in a given 2D integer array. The 2D successor array has the same dimensions as the given 2D integer array. Each element in the 2D successor array is the position (row, column) of the corresponding 2D integer array element's successor. The largest element in the 2D integer array does not have a successor in the 2D integer array, so its corresponding position in the 2D successor array is `null`.

The following diagram shows a 2D integer array and its corresponding 2D successor array. To illustrate the successor relationship, the values 8 and 9 in the 2D integer array are shaded. In the 2D successor array, the shaded element shows that the position of the successor of 8 is `(0,2)` in the 2D integer array. The largest value in the 2D integer array is 16, so its corresponding element in the 2D successor array is `null`.

2D Integer Array

	0	1	2	3
0	15	5	9	10
1	12	16	11	6
2	14	8	13	7

2D Successor Array

	0	1	2	3
0	(1,1)	(1,3)	(0,3)	(1,2)
1	(2,2)	null	(1,0)	(2,3)
2	(0,0)	(0,2)	(2,0)	(2,1)

Class information for this question

```
public class Position

public Position(int r, int c)

public class Successors

public static Position findPosition(int num, int[][] intArr)
public static Position[][] getSuccessorArray(int[][] intArr)
```

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
```

STOP

END OF EXAM

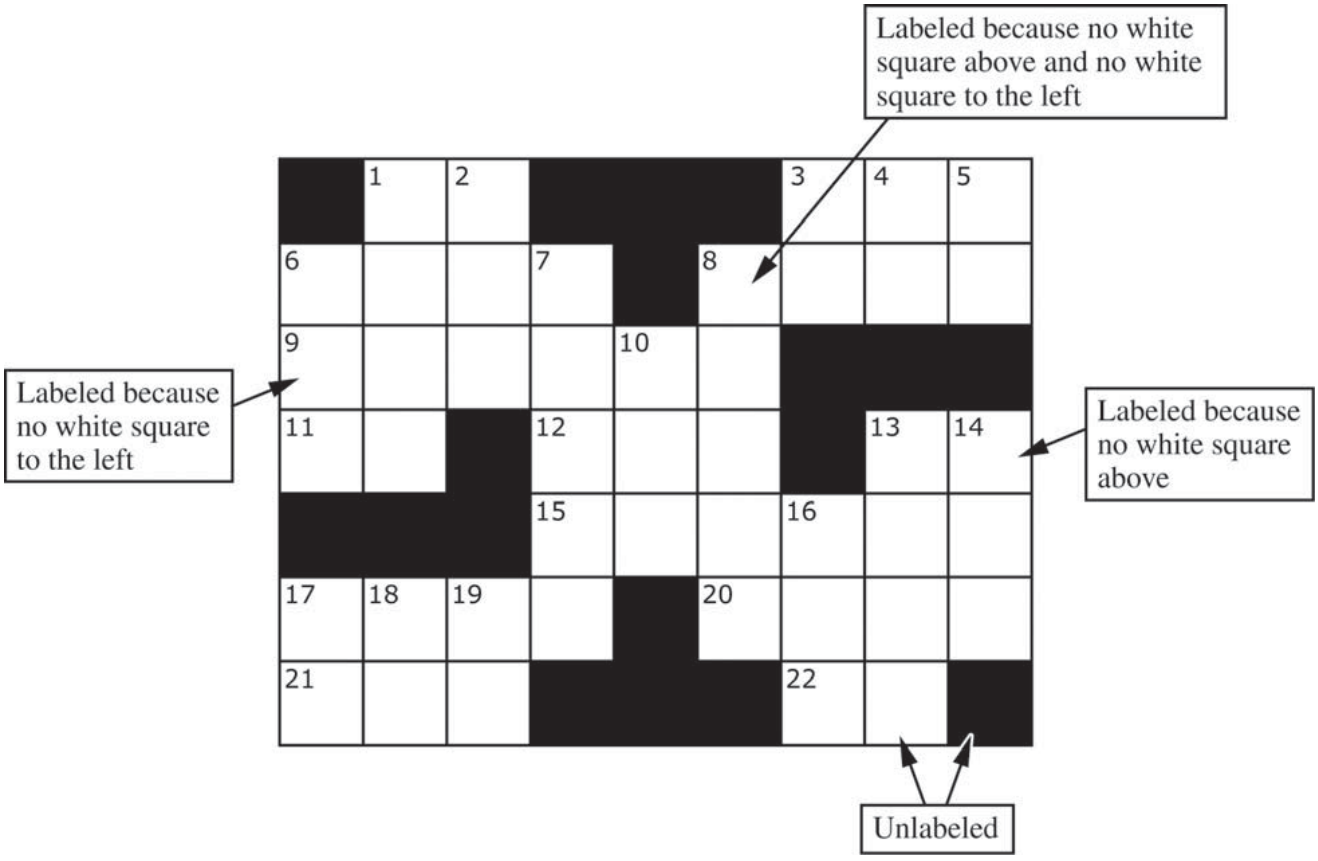
3. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

   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.



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.

```
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.
}
```

Part (a) begins on page 14.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (b) begins on page 16.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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

Notes:
- Assume that the classes listed in the Java 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 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`.

	`arr1`				Value returned by `arraySum(arr1)`
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

(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)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (c) begins on page 6.

(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)
```

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

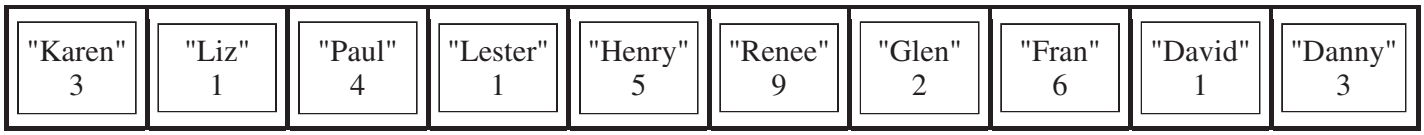
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** 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<Student> roster` contains references to `Student` objects in the following order.



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`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Part (a) continues on page 9.

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)
```

Part (b) begins on page 10.

(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)
```

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.14 Searching Algorithms

Name: _____ Class: _____ Date: _____ **Total: 8 marks**

KEY WORDS binary search 二分查找 • linear search 线性查找

Objective

Build the skills to answer exam questions on **searching algorithms**.

You must be able to:

- perform a **linear search** 线性查找 by checking each element in turn
- perform a **binary search** 二分查找 on a **sorted** collection
- explain why binary search requires sorted data

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Linear search

Check each element in turn until a match is found (or the end is reached). It works on **any** list, sorted or not.

■ Binary search

On a **sorted** list, look at the middle; if the target is smaller, search the left half, else the right — each step **halves** the range. It is far faster (about $\log_2 n$ steps), but the data **must be sorted first**.

2 Practice

2.1 Describe a linear search. [1]

2.2 State the requirement for binary search to work. [1]

2.3 State which search is faster on a large sorted list. [1]

3 Exam-style questions

3.1 A linear search checks [1]

- **A** the middle element only
- **B** each element in turn
- **C** random elements
- **D** no elements

3.2 Binary search requires the data to be [1]

- **A** reversed
- **B** sorted
- **C** empty
- **D** in random order

3.3 A sorted list has 32 items.

(a) Name the faster search method. [1]

(b) State roughly how many steps binary search needs ($32 \to 16 \to 8 \to 4 \to 2 \to 1$). [1]

(c) Name the search that works on unsorted data. [1]

Solutions

2.1 checking each element one by one until a match is found.

2.2 the data must be sorted.

2.3 binary search.

3.1 B.

3.2 B.

3.3 (a) binary search. (b) about 5. (c) linear search.

4. This question involves reasoning about a number puzzle that is represented as a two-dimensional array of integers. Each element of the array initially contains a value between 1 and 9, inclusive. Solving the puzzle involves clearing pairs of array elements by setting them to 0. Two elements can be cleared if their values sum to 10 or if they have the same value. The puzzle is considered solved if all elements of the array are cleared.

You will write the constructor and one method of the `SumOrSameGame` class, which contains the methods that manipulate elements of the puzzle.

```
public class SumOrSameGame
{
  private int[][] puzzle;

  /**
   * Creates a two-dimensional array and fills it with random integers,
   * as described in part (a)
   * Precondition: numRows > 0; numCols > 0
   */
  public SumOrSameGame(int numRows, int numCols)
  { /* to be implemented in part (a) */ }

  /**
   * Identifies and clears an element of puzzle that can be paired with
   * the element at the given row and column, as described in part (b)
   * Preconditions: row and col are valid row and column indices in puzzle.
   *  The element at the given row and column is between 1 and 9, inclusive.
   */
  public boolean clearPair(int row, int col)
  { /* to be implemented in part (b) */ }

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

A. Write the constructor for the `SumOrSameGame` class. The constructor initializes the instance variable `puzzle` to be a two-dimensional integer array with the number of rows and columns specified by the parameters `numRows` and `numCols`, respectively. Array elements are initialized with random integers between `1` and `9`, inclusive, each with an equal chance of being assigned to each element of `puzzle`.

When an element of the two-dimensional array is accessed, the first index is used to specify the row and the second index is used to specify the column.

Complete the `SumOrSameGame` constructor.

```
/**
 * Creates a two-dimensional array and fills it with random integers,
 * as described in part (a)
 * Precondition: numRows > 0; numCols > 0
 */
public SumOrSameGame(int numRows, int numCols)
```

B. Write the `clearPair` method, which takes a valid row index and valid column index as its parameters. The array element specified by those indices, which has a value between `1` and `9`, inclusive, is compared to other array elements in `puzzle` in an attempt to pair it with another array element that meets both of the following conditions.

- The row index of the second element is greater than or equal to the parameter `row`.
- The two elements have equal values or have values that sum to 10.

If such an array element is found, both array elements of the pair are cleared (set to `0`) and the method returns `true`. If more than one such array element is found, any one of those identified array elements can be used to complete the pair and can be cleared. If no such array element is found, no changes are made to `puzzle` and the method returns `false`.

The following table shows the possible results of several calls to `clearPair`.

`puzzle` before call to `clearPair`	Method Call	`puzzle` after call to `clearPair`	Return Value	Explanation

`0 7 9 0`
`7 4 1 6`
`8 4 1 8`
	`clearPair(0, 1)`	`0 0 9 0`
		`0 4 1 6`
		`8 4 1 8`
			`true`	The value `7` in row `0`, column `1` is matched with the value `7` in row `1`, column `0`.

`1 2 3 4`
`5 6 7 8`
`5 4 1 2`
	`clearPair(2, 2)`	`1 2 3 4`
		`5 6 7 8`
		`5 4 1 2`
			`false`	There is no element in row `2` or a later row to pair with the value `1`.

`8 1 0 5`
`0 4 3 6`
`3 4 5 8`
	`clearPair(1, 1)`	`8 1 0 5`
		`0 0 3 0`
		`3 4 5 8`
			`true`	The value `4` in row `1`, column `1` is matched with the value `6` in row `1`, column `3`. It could also have been matched with the value `4` in row `2`, column `1`.

`1 7 9`
`2 6 5`
`4 4 4`
	`clearPair(0, 2)`	`0 7 0`
		`2 6 5`
		`4 4 4`
			`true`	The value `9` in row `0`, column `2` is matched with the value `1` in row `0`, column `0`.

Complete method `clearPair`.

```
/**
 * Identifies and clears an element of puzzle that can be paired with
 * the element at the given row and column, as described in part (b)
 * Preconditions: row and col are valid row and column indices in puzzle.
 *  The element at the given row and column is between 1 and 9, inclusive.
 */
public boolean clearPair(int row, int col)
```

STOP

END OF EXAM

1. This question involves the `AppointmentBook` class, which provides methods for students to schedule appointments with their teacher. Appointments can be scheduled during one of eight class periods during the school day, numbered 1 through 8. A requested appointment has a duration, which is the number of minutes the appointment will last. The 60 minutes within a period are numbered 0 through 59. In order for an appointment to be scheduled, the teacher must have a block of consecutive, available minutes that contains at least the requested number of minutes in a requested period. Scheduled appointments must start and end within the same period.

The `AppointmentBook` class contains two helper methods, `isMinuteFree` and `reserveBlock`. You will write two additional methods of the `AppointmentBook` class.

```
public class AppointmentBook
{
   /**
    *  Returns true if minute in period is available for an appointment and returns
    *  false otherwise
    *  Preconditions: 1 <= period <= 8;  0 <= minute <= 59
    */
   private boolean isMinuteFree(int period, int minute)
   {  /* implementation not shown */  }

   /**
    *  Marks the block of minutes that starts at startMinute in period and
    *  is duration minutes long as reserved for an appointment
    *  Preconditions: 1 <= period <= 8; 0 <= startMinute <= 59;
    *      1 <= duration <= 60
    */
   private void reserveBlock(int period, int startMinute, int duration)
   {  /* implementation not shown */  }

   /**
    *  Searches for the first block of duration free minutes during period, as described in
    *  part (a). Returns the first minute in the block if such a block is found or returns -1 if no
    *  such block is found.
    *  Preconditions: 1 <= period <= 8; 1 <= duration <= 60
    */
   public int findFreeBlock(int period, int duration)
   {  /* to be implemented in part (a) */  }

   /**
    *  Searches periods from startPeriod to endPeriod, inclusive, for a block
    *  of duration free minutes, as described in part (b). If such a block is found,
    *  calls reserveBlock to reserve the block of minutes and returns true; otherwise
    *  returns false.
    *  Preconditions: 1 <= startPeriod <= endPeriod <= 8; 1 <= duration <= 60
    */
   public boolean makeAppointment(int startPeriod, int endPeriod,
                                  int duration)
   {  /* to be implemented in part (b) */  }

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

(a) Write the `findFreeBlock` method, which searches `period` for the first block of free minutes that is `duration` minutes long. If such a block is found, `findFreeBlock` returns the first minute in the block. Otherwise, `findFreeBlock` returns `-1`. The `findFreeBlock` method uses the helper method `isMinuteFree`, which returns `true` if a particular minute is available to be included in a new appointment and returns `false` if the minute is unavailable.

Consider the following list of unavailable and available minutes in period 2.

Minutes in Period 2	Available?
0–9 (10 minutes)	No
10–14 (5 minutes)	Yes
15–29 (15 minutes)	No
30–44 (15 minutes)	Yes
45–49 (5 minutes)	No
50–59 (10 minutes)	Yes

The method call `findFreeBlock(2, 15)` would return `30` to indicate that a 15-minute block starting with minute `30` is available. No steps should be taken as a result of the call to `findFreeBlock` to mark those 15 minutes as unavailable.

The method call `findFreeBlock(2, 9)` would also return `30`. Whenever there are multiple blocks that satisfy the requirement, the earliest starting minute is returned.

The method call `findFreeBlock(2, 20)` would return `-1`, since no 20-minute block of available minutes exists in period 2.

Complete method `findFreeBlock`. You must use `isMinuteFree` appropriately in order to receive full credit.

```
/**
 *  Searches for the first block of duration free minutes during period, as described in
 *  part (a). Returns the first minute in the block if such a block is found or returns -1 if no
 *  such block is found.
 *  Preconditions: 1 <= period <= 8; 1 <= duration <= 60
 */
public int findFreeBlock(int period, int duration)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `makeAppointment` method, which searches the periods from `startPeriod` to `endPeriod`, inclusive, for the earliest block of `duration` available minutes in the lowest-numbered period. If such a block is found, the `makeAppointment` method calls the helper method `reserveBlock` to mark the minutes in the block as unavailable and returns `true`. If no such block is found, the `makeAppointment` method returns `false`.

Consider the following list of unavailable and available minutes in periods 2, 3, and 4 and three successive calls to `makeAppointment`.

Period	Minutes	Available?
2	0–24 (25 minutes)	No
2	25–29 (5 minutes)	Yes
2	30–59 (30 minutes)	No
3	0–14 (15 minutes)	Yes
3	15–40 (26 minutes)	No
3	41–59 (19 minutes)	Yes
4	0–4 (5 minutes)	No
4	5–29 (25 minutes)	Yes
4	30–43 (14 minutes)	No
4	44–59 (16 minutes)	Yes

The method call `makeAppointment(2, 4, 22)` returns `true` and results in the minutes 5 through 26, inclusive, in period 4 being marked as unavailable.

The method call `makeAppointment(3, 4, 3)` returns `true` and results in the minutes 0 through 2, inclusive, in period 3 being marked as unavailable.

The method call `makeAppointment(2, 4, 30)` returns `false`, since there is no block of 30 available minutes in periods 2, 3, or 4.

The following shows the updated list of unavailable and available minutes in periods 2, 3, and 4 after the three example method calls are complete.

Period	Minutes	Available?
2	0–24 (25 minutes)	No
2	25–29 (5 minutes)	Yes
2	30–59 (30 minutes)	No
3	0–2 (3 minutes)	No
3	3–14 (12 minutes)	Yes
3	15–40 (26 minutes)	No
3	41–59 (19 minutes)	Yes
4	0–26 (27 minutes)	No
4	27–29 (3 minutes)	Yes
4	30–43 (14 minutes)	No
4	44–59 (16 minutes)	Yes

Complete method `makeAppointment`. Assume that `findFreeBlock` works as intended, regardless of what you wrote in part (a). You must use `findFreeBlock` and `reserveBlock` appropriately in order to receive full credit.

```
/**
 *  Searches periods from startPeriod to endPeriod, inclusive, for a block
 *  of duration free minutes, as described in part (b). If such a block is found,
 *  calls reserveBlock to reserve the block of minutes and returns true; otherwise
 *  returns false.
 *  Preconditions: 1 <= startPeriod <= endPeriod <= 8; 1 <= duration <= 60
 */
public boolean makeAppointment(int startPeriod, int endPeriod,
                               int duration)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class AppointmentBook

private boolean isMinuteFree(int period, int minute)
private void reserveBlock(int period, int startMinute,
   int duration)
public int findFreeBlock(int period, int duration)
public boolean makeAppointment(int startPeriod, int endPeriod,
   int duration)
```

4. This question involves pieces of candy in a box. The `Candy` class represents a single piece of candy.

```
public class Candy
{
    /** Returns a String representing the flavor of this piece of candy */
    public String getFlavor()
    {  /* implementation not shown */  }

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

The `BoxOfCandy` class represents a candy box where the candy is arranged in a rectangular grid. The instance variable of the class, `box`, is a rectangular two-dimensional array of `Candy` objects. A location in the candy box may contain a piece of candy or may be empty. A piece of candy is represented by a `Candy` object. An empty location is represented by `null`.

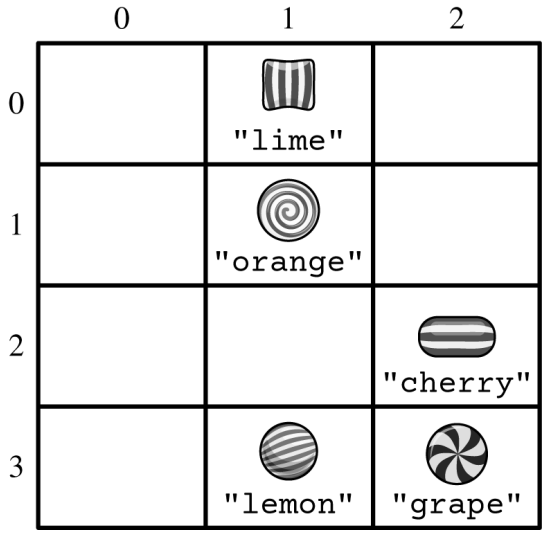
You will write two methods of the `BoxOfCandy` class.

```
public class BoxOfCandy
{
    /** box contains at least one row and is initialized in the constructor. */
    private Candy[][] box;

    /**
     *  Moves one piece of candy in column col, if necessary and possible, so that the box
     *  element in row 0 of column col contains a piece of candy, as described in part (a).
     *  Returns false if there is no piece of candy in column col and returns true otherwise.
     *  Precondition: col is a valid column index in box.
     */
    public boolean moveCandyToFirstRow(int col)
    {  /* to be implemented in part (a) */  }

    /**
     *  Removes from box and returns a piece of candy with flavor specified by the parameter, or
     *  returns null if no such piece is found, as described in part (b)
     */
    public Candy removeNextByFlavor(String flavor)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `moveCandyToFirstRow` method, which attempts to ensure that the `box` element at row `0` and column `col` contains a piece of candy, using the following steps.

- If the element at row `0` and column `col` already contains a piece of candy, then `box` is unchanged and the method returns `true`.
- If the element at row `0` and column `col` does not contain a piece of candy, then the method searches the remaining rows of column `col` for a piece of candy. If a piece of candy can be found in column `col`, it is moved to row `0`, its previous location is set to `null`, and the method returns `true`; otherwise, the method returns `false`.

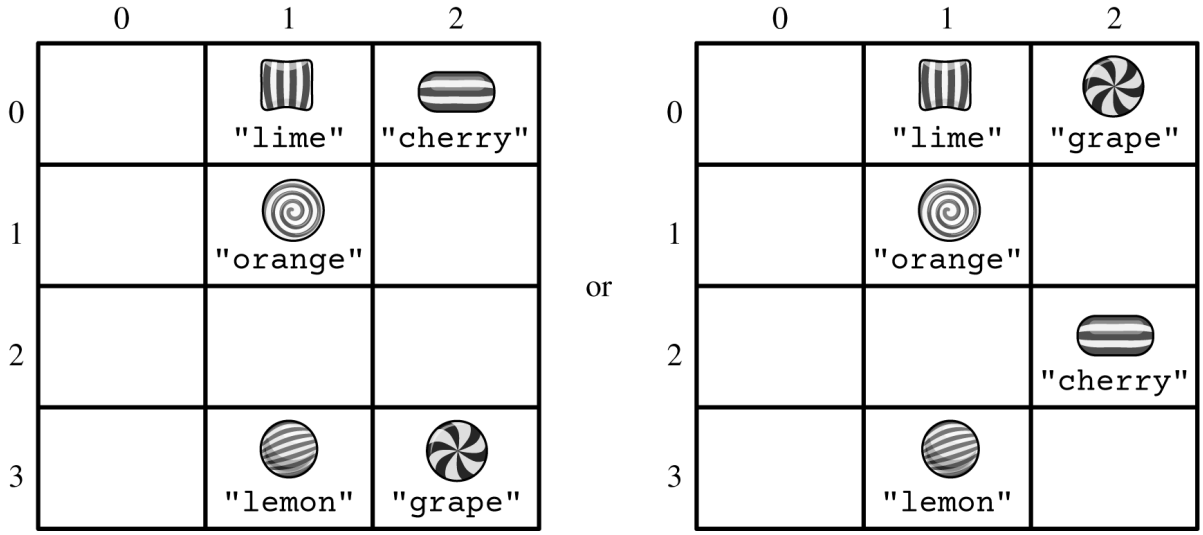
In the following example, the grid represents the contents of `box`. An empty square in the grid is `null` in `box`. A non-empty square in the grid represents a `box` element that contains a `Candy` object. The string in the square of the grid indicates the flavor of the piece of candy.



The method call `moveCandyToFirstRow(0)` returns `false` because the `box` element at row `0` and column `0` does not contain a piece of candy and there are no pieces of candy in column `0` that can be moved to row `0`. The contents of `box` are unchanged.

The method call `moveCandyToFirstRow(1)` returns `true` because the `box` element at row `0` and column `1` already contains a piece of candy. The contents of `box` are unchanged.

The method call `moveCandyToFirstRow(2)` moves one of the two pieces of candy in column `2` to row `0` of column `2`, sets the previous location of the piece of candy that was moved to `null`, and returns `true`. The new contents of `box` could be either of the following.



Complete the `moveCandyToFirstRow` method.

```
/**
 * Moves one piece of candy in column col, if necessary and possible, so that the box
 * element in row 0 of column col contains a piece of candy, as described in part (a).
 * Returns false if there is no piece of candy in column col and returns true otherwise.
 * Precondition: col is a valid column index in box.
 */
public boolean moveCandyToFirstRow(int col)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

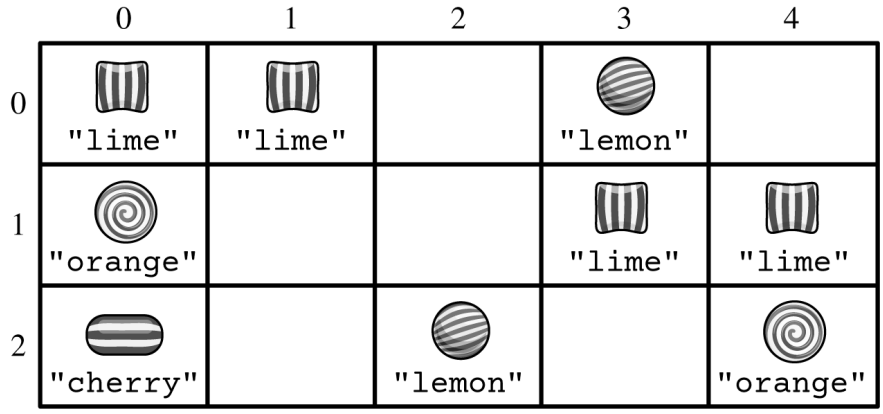
```
public class Candy
public String getFlavor()
public class BoxOfCandy
private Candy[][] box
public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

(b) Write the `removeNextByFlavor` method, which attempts to remove and return one piece of candy from the box. The piece of candy to be removed is the first piece of candy with a flavor equal to the parameter `flavor` that is encountered while traversing the candy box in the following order: the last row of the box is traversed from left to right, then the next-to-last row of the box is traversed from left to right, etc., until either a piece of candy with the desired flavor is found or until the entire candy box has been searched.

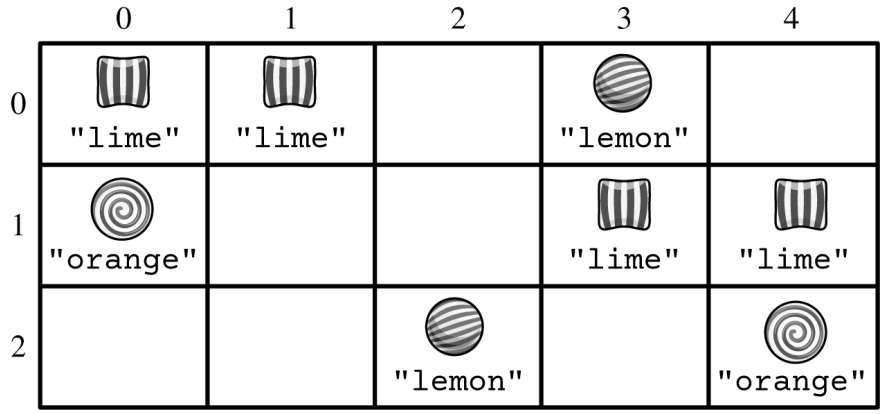
If the `removeNextByFlavor` method finds a `Candy` object with the desired flavor, the corresponding `box` element is assigned `null`, all other `box` elements are unchanged, and the removed `Candy` object is returned. Otherwise, `box` is unchanged and the method returns `null`.

The following examples show three consecutive calls to the `removeNextByFlavor` method. The traversal of the candy box always begins in the last row and first column of the box.

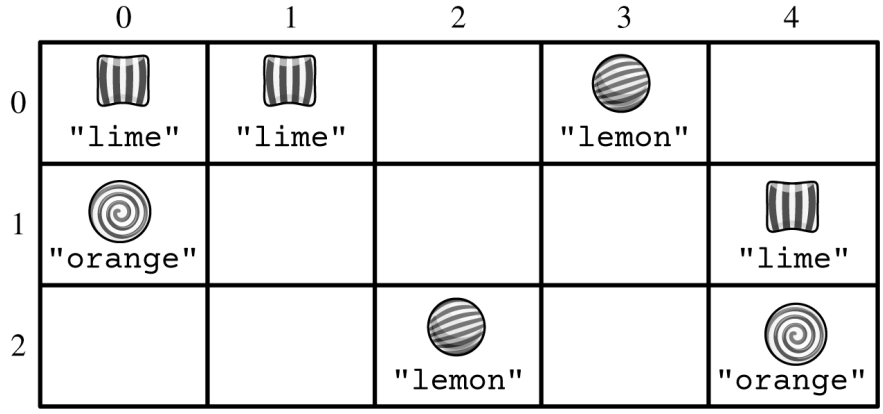
The following grid shows the contents of `box` before any of the `removeNextByFlavor` method calls.



The method call `removeNextByFlavor("cherry")` removes and returns the `Candy` object located in row `2` and column `0`. The following grid shows the updated contents of `box`.



The method call `removeNextByFlavor("lime")` removes and returns the `Candy` object located in row `1` and column `3`. The following grid shows the updated contents of `box`.



The method call `removeNextByFlavor("grape")` returns `null` because no grape-flavored candy is found. The contents of `box` are unchanged.

Complete the `removeNextByFlavor` method.

```
/**
 *  Removes from box and returns a piece of candy with flavor specified by the parameter, or
 *  returns null if no such piece is found, as described in part (b)
 */
public Candy removeNextByFlavor(String flavor)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Candy

public String getFlavor()

public class BoxOfCandy

private Candy[][] box

public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

STOP

END OF EXAM

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.15 Sorting Algorithms

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS selection sort 选择排序 • insertion sort 插入排序 • array 数组 • 2d array 二维数组

Objective

Build the skills to answer exam questions on **sorting algorithms**.

You must be able to:

- describe how **selection sort** 选择排序 moves the smallest remaining value into place
- describe how **insertion sort** 插入排序 builds a sorted section one element at a time
- trace a collection after each pass of a sort

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Selection sort

Repeatedly find the **smallest remaining** value and move it into its place at the front of the unsorted part.

■ Insertion sort

Build a **sorted section** at the front, one element at a time, inserting each new value into its correct spot.

■ Tracing (selection sort on `{3, 1, 2}`)

- Pass 1: smallest is 1 → `{1, 3, 2}`
- Pass 2: smallest of the rest is 2 → `{1, 2, 3}`

2 Practice

2.1 Describe how selection sort works. [1]

2.2 Describe how insertion sort works. [1]

2.3 After one pass of selection sort on `{5, 2, 8, 1}`, state the array. [2]

3 Exam-style questions

3.1 Selection sort repeatedly moves the _____ remaining value into place. [1]

- **A** largest
- **B** smallest
- **C** middle
- **D** random

3.2 Insertion sort builds a [1]

- **A** random section
- **B** sorted section one element at a time
- **C** 2D array
- **D** linked list

3.3 Apply selection sort to `{4, 1, 3, 2}` (smallest moved to the front each pass).

(a) State the array after pass 1. [1]

(b) State the array after pass 2. [1]

(c) State the fully sorted result. [1]

Solutions

2.1 it repeatedly finds the smallest remaining value and moves it into place at the front of the unsorted part.

2.2 it builds a sorted section one element at a time, inserting each new value in order.

2.3 `{1, 2, 8, 5}` — the smallest (1) is moved to the front.

3.1 B.

3.2 B.

3.3 (a) `{1, 4, 3, 2}`. (b) `{1, 2, 3, 4}`. (c) `{1, 2, 3, 4}`.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.16 Recursion

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS base case 基准情形 • recursion 递归 • recursive case 递归情形

Objective

Build the skills to answer exam questions on **recursion**.

You must be able to:

- understand that a **recursive** 递归 method calls itself
- identify the **base case** 基准情形 that stops the recursion
- identify the **recursive case** 递归情形 that moves toward the base case

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Recursion

A **recursive** method calls **itself**. It needs:

- a **base case** that stops the recursion, and
- a **recursive case** that moves the problem toward the base case.

```
int fact(int n) {
    if (n <= 1) return 1;      // base case
    return n * fact(n - 1);    // recursive case
}
```

Without a base case, the recursion never stops.

■ Unwinding the calls

`fact(3)` calls `fact(2)`, which calls `fact(1)`. The base case returns `1`, then the waiting calls finish in reverse: `fact(2) = 2 * 1 = 2`, then `fact(3) = 3 * 2 = 6`. Each call pauses until the one below it returns.

2 Practice

2.1 Define a recursive method. [1]

2.2 State the purpose of a base case. [1]

2.3 For the `fact` example above, state the value of `fact(3)`. [2]

3 Exam-style questions

3.1 A recursive method is one that [1]

- **A** never returns
- **B** calls itself
- **C** always uses a loop
- **D** must be `static`

3.2 The part that stops the recursion is the [1]

- **A** recursive case
- **B** base case
- **C** parameter
- **D** return type

3.3 `int f(int n) { if (n == 0) return 0; return n + f(n - 1); }`.

(a) State `f(0)`. [1]

(b) State `f(3)`. [1]

(c) Name the case where `n == 0`. [1]

Solutions

2.1 a method that calls itself.

2.2 it stops the recursion so it does not run forever.

2.3 $3 \times 2 \times 1 = 6$.

3.1 B.

3.2 B.

3.3 (a) `0`. (b) $3 + 2 + 1 + 0 = 6$. (c) the base case.

AP COMPUTER SCIENCE A • EXERCISE SHEET

4.17 Recursive Searching and Sorting

Name: _____ Class: _____ Date: _____ **Total: 9 marks**

KEY WORDS merge sort 归并排序 • binary search 二分搜索

Objective

Build the skills to answer exam questions on **recursive searching and sorting**.

You must be able to:

- describe how **binary search** can be written as a **recursive** method
- trace a recursive binary search halving the range on each call
- describe how **merge sort** 归并排序 splits, sorts, and merges

1 Worked examples

Study these first. Each one shows the method for a question type used later.

■ Recursive binary search

Binary search is naturally recursive: check the **middle**; if the target is smaller, **recurse on the left half**, else recurse on the right — each call halves the range.

■ Merge sort

1. **Split** the collection into two halves.
2. **Sort each half** (recursively, using merge sort).
3. **Merge** the two sorted halves into one sorted collection.

2 Practice

2.1 State how binary search can be written recursively. [1]

2.2 State the three steps of merge sort. [2]

2.3 State what merge sort does after sorting the two halves. [1]

3 Exam-style questions

3.1 Merge sort works by [1]

- **A** checking each element in turn
- **B** splitting, sorting each half, and merging
- **C** swapping neighbouring elements only
- **D** not sorting at all

3.2 A recursive binary search calls itself on [1]

- **A** the whole list again
- **B** the correct half of the list
- **C** a random half
- **D** nothing

3.3 Merge sort is applied to a list.

(a) State the first step. [1]

(b) State what happens to each half. [1]

(c) State the final step. [1]

Solutions

2.1 it checks the middle element, then calls itself on the half that could contain the target.

2.2 split into two halves; sort each half; merge them.

2.3 it merges the two sorted halves into one sorted list.

3.1 B.

3.2 B.

3.3 (a) split the list into two halves. (b) each half is sorted (recursively). (c) merge the two sorted halves.