2021

AP CollegeBoard

AP® Computer Science A

Free-Response Questions

COMPUTER SCIENCE A

SECTION II

Time—1 hour and 30 minutes

4 Questions

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA. You may plan your answers in this orange booklet, but no credit will be given for anything written in this booklet. **You will only earn credit for what you write in the separate Free Response booklet.**

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.

GO ON TO THE NEXT PAGE.

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
   /** The secret string. */
   private String secret;

   /** Constructs a WordMatch object with the given secret string of lowercase letters. */
   public WordMatch(String word)
   {
      /* implementation not shown */
   }

   /** Returns a score for guess, as described in part (a).
    *  Precondition: 0 < guess.length() <= secret.length()
    */
   public int scoreGuess(String guess)
   {  /* to be implemented in part (a) */  }

   /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
    *  tie-breaker that are described in part (b).
    *  Precondition: guess1 and guess2 contain all lowercase letters.
    *                guess1 is not the same as guess2.
    */
   public String findBetterGuess(String guess1, String guess2)
   {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"i"`	4	`4 * 1 * 1 = 4`	4
`"iss"`	2	`2 * 3 * 3 = 18`	18
`"issipp"`	1	`1 * 6 * 6 = 36`	36
`"mississippi"`	1	`1 * 11 * 11 = 121`	121

```
WordMatch game = new WordMatch("aaaabb");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: (**Number of Substring Occurrences**) x (**Square of the Length of** `guess`)	**Return Value of** `game.scoreGuess(guess)`
`"a"`	4	`4 * 1 * 1 = 4`	4
`"aa"`	3	`3 * 2 * 2 = 12`	12
`"aaa"`	2	`2 * 3 * 3 = 18`	18
`"aabb"`	1	`1 * 4 * 4 = 16`	16
`"c"`	0	`0 * 1 * 1 = 0`	0

GO ON TO THE NEXT PAGE.

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).
 *  Precondition: 0 < guess.length() <= secret.length()
 */
public int scoreGuess(String guess)
```

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 WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten");`	`9`	`1 * 3 * 3`
`game.scoreGuess("nation");`	`36`	`1 * 6 * 6`
`game.findBetterGuess("ten", "nation");`	`"nation"`	Since `scoreGuess` returns `36` for `"nation"` and `9` for `"ten"`, the guess with the greater score, `"nation"`, is returned.
`game.scoreGuess("con");`	`9`	`1 * 3 * 3`
`game.scoreGuess("cat");`	`9`	`1 * 3 * 3`
`game.findBetterGuess("con", "cat");`	`"con"`	Since `scoreGuess` returns `9` for both `"con"` and `"cat"`, the alphabetically greater guess, `"con"`, is returned.

GO ON TO THE NEXT PAGE.

Complete method `findBetterGuess`.

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

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 *  tie-breaker that are described in part (b).
 *  Precondition: guess1 and guess2 contain all lowercase letters.
 *                guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

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 WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

GO ON TO THE NEXT PAGE.

2. The class `SingleTable` represents a table at a restaurant.

```
public class SingleTable
{
    /** Returns the number of seats at this table. The value is always greater than or equal to 4. */
    public int getNumSeats()
    {  /* implementation not shown */  }

    /** Returns the height of this table in centimeters. */
    public int getHeight()
    {  /* implementation not shown */  }

    /** Returns the quality of the view from this table. */
    public double getViewQuality()
    {  /* implementation not shown */  }

    /** Sets the quality of the view from this table to value. */
    public void setViewQuality(double value)
    {  /* implementation not shown */  }

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

At the restaurant, customers can sit at tables that are composed of two single tables pushed together. You will write a class `CombinedTable` to represent the result of combining two `SingleTable` objects, based on the following rules and the examples in the chart that follows.

- A `CombinedTable` can seat a number of customers that is two fewer than the total number of seats in its two `SingleTable` objects (to account for seats lost when the tables are pushed together).
- A `CombinedTable` has a desirability that depends on the views and heights of the two single tables. If the two single tables of a `CombinedTable` object are the same height, the desirability of the `CombinedTable` object is the average of the view qualities of the two single tables.
- If the two single tables of a `CombinedTable` object are not the same height, the desirability of the `CombinedTable` object is 10 units less than the average of the view qualities of the two single tables.

GO ON TO THE NEXT PAGE.

Assume `SingleTable` objects `t1`, `t2`, and `t3` have been created as follows.

- `SingleTable t1` has 4 seats, a view quality of 60.0, and a height of 74 centimeters.
- `SingleTable t2` has 8 seats, a view quality of 70.0, and a height of 74 centimeters.
- `SingleTable t3` has 12 seats, a view quality of 75.0, and a height of 76 centimeters.

The chart contains a sample code execution sequence and the corresponding results.

Statement	Value Returned (blank if no value)	Class Specification
`CombinedTable c1 = new CombinedTable(t1, t2);`		A `CombinedTable` is composed of two `SingleTable` objects.
`c1.canSeat(9);`	`true`	Since its two single tables have a total of 12 seats, `c1` can seat 10 or fewer people.
`c1.canSeat(11);`	`false`	`c1` cannot seat 11 people.
`c1.getDesirability();`	`65.0`	Because `c1`'s two single tables are the same height, its desirability is the average of `60.0` and `70.0`.
`CombinedTable c2 = new CombinedTable(t2, t3);`		A `CombinedTable` is composed of two `SingleTable` objects.
`c2.canSeat(18);`	`true`	Since its two single tables have a total of 20 seats, `c2` can seat 18 or fewer people.
`c2.getDesirability();`	`62.5`	Because `c2`'s two single tables are not the same height, its desirability is 10 units less than the average of `70.0` and `75.0`.
`t2.setViewQuality(80);`		Changing the view quality of one of the tables that makes up `c2` changes the desirability of `c2`, as illustrated in the next line of the chart. Since `setViewQuality` is a `SingleTable` method, you do not need to write it.
`c2.getDesirability();`	`67.5`	Because the view quality of `t2` changed, the desirability of `c2` has also changed.

The last line of the chart illustrates that when the characteristics of a `SingleTable` change, so do those of the `CombinedTable` that contains it.

Write the complete `CombinedTable` class. Your implementation must meet all specifications and conform to the examples shown in the preceding chart.

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.

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.
}
```

GO ON TO THE NEXT PAGE.

(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

GO ON TO THE NEXT PAGE.

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)
```

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) */  }
}
```

GO ON TO THE NEXT PAGE.

(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)
```

GO ON TO THE NEXT PAGE.

STOP

END OF EXAM