

**2022**

**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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

1. This question involves simulation of the play and scoring of a single-player video game. In the game, a player attempts to complete three levels. A level in the game is represented by the `Level` class.

```
public class Level
{
    /** Returns true if the player reached the goal on this level and returns false otherwise */
    public boolean goalReached()
    { /* implementation not shown */ }

    /** Returns the number of points (a positive integer) recorded for this level */
    public int getPoints()
    { /* implementation not shown */ }

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

**GO ON TO THE NEXT PAGE.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

Play of the game is represented by the Game class. You will write two methods of the Game class.

```
public class Game
{
    private Level levelOne;
    private Level levelTwo;
    private Level levelThree;

    /** Postcondition: All instance variables have been initialized. */
    public Game()
    { /* implementation not shown */ }

    /** Returns true if this game is a bonus game and returns false otherwise */
    public boolean isBonus()
    { /* implementation not shown */ }

    /** Simulates the play of this Game (consisting of three levels) and updates all relevant
     *  game data
     */
    public void play()
    { /* implementation not shown */ }

    /** Returns the score earned in the most recently played game, as described in part (a) */
    public int getScore()
    { /* to be implemented in part (a) */ }

    /** Simulates the play of num games and returns the highest score earned, as
     *  described in part (b)
     *  Precondition: num > 0
     */
    public int playManyTimes(int num)
    { /* to be implemented in part (b) */ }

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

**GO ON TO THE NEXT PAGE.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (a) Write the `getScore` method, which returns the score for the most recently played game. Each game consists of three levels. The score for the game is computed using the following helper methods.

- The `isBonus` method of the `Game` class returns `true` if this is a bonus game and returns `false` otherwise.
- The `goalReached` method of the `Level` class returns `true` if the goal has been reached on a particular level and returns `false` otherwise.
- The `getPoints` method of the `Level` class returns the number of points recorded on a particular level. Whether or not recorded points are earned (included in the game score) depends on the rules of the game, which follow.

The score for the game is computed according to the following rules.

- Level one points are earned only if the level one goal is reached. Level two points are earned only if both the level one and level two goals are reached. Level three points are earned only if the goals of all three levels are reached.
- The score for the game is the sum of the points earned for levels one, two, and three.
- If the game is a bonus game, the score for the game is tripled.

**GO ON TO THE NEXT PAGE.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

The following table shows some examples of game score calculations.

	<b>Level One Results</b>	<b>Level Two Results</b>	<b>Level Three Results</b>	<b>isBonus Return Value</b>	<b>Score Calculation</b>
<b>goalReached Return Value:</b>  <b>getPoints Return Value:</b>	true  200	true  100	true  500	true	$(200 + 100 + 500) \times 3 = 2,400$ The recorded points for levels one, two, and three are earned because the goals were reached in all three levels. The earned points are multiplied by 3 because isBonus returns true.
<b>goalReached Return Value:</b>  <b>getPoints Return Value:</b>	true  200	true  100	false  500	false	$200 + 100 = 300$ The recorded points for level one and level two are earned because the goal was reached in levels one and two. The recorded points for level three are not earned because the goal was not reached in level three.
<b>goalReached Return Value:</b>  <b>getPoints Return Value:</b>	true  200	false  100	true  500	true	$200 \times 3 = 600$ The recorded points for only level one are earned because the goal was not reached in level two. The earned points are multiplied by 3 because isBonus returns true.
<b>goalReached Return Value:</b>  <b>getPoints Return Value:</b>	false  200	true  100	true  500	false	0 Because the goal in level one was not reached, no points are earned for any level.

Complete the getScore method.

```
/** Returns the score earned in the most recently played game, as described in part (a) */
public int getScore()
```

---

**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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (b) Write the `playManyTimes` method, which simulates the play of `num` games and returns the highest game score earned. For example, if the four plays of the game that are simulated as a result of the method call `playManyTimes(4)` earn scores of 75, 50, 90, and 20, then the method should return 90.

Play of the game is simulated by calling the helper method `play`. Note that if `play` is called only one time followed by multiple consecutive calls to `getScore`, each call to `getScore` will return the score earned in the single simulated play of the game.

Complete the `playManyTimes` method. Assume that `getScore` works as intended, regardless of what you wrote in part (a). You must call `play` and `getScore` appropriately in order to receive full credit.

```
/** Simulates the play of num games and returns the highest score earned, as
 * described in part (b)
 * Precondition: num > 0
 */
public int playManyTimes(int num)
```

**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 Level
public boolean goalReached()
public int getPoints()

public class Game
private Level levelOne
private Level levelTwo
private Level levelThree

public Game()
public boolean isBonus()
public void play()
public int getScore()
public int playManyTimes(int num)
```

**GO ON TO THE NEXT PAGE.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

2. The Book class is used to store information about a book. A partial Book class definition is shown.

```

public class Book
{
    /** The title of the book */
    private String title;

    /** The price of the book */
    private double price;

    /** Creates a new Book with given title and price */
    public Book(String bookTitle, double bookPrice)
    { /* implementation not shown */ }

    /** Returns the title of the book */
    public String getTitle()
    { return title; }

    /** Returns a string containing the title and price of the Book */
    public String getBookInfo()
    {
        return title + " - " + price;
    }

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

```

You will write a class Textbook, which is a subclass of Book.

A Textbook has an edition number, which is a positive integer used to identify different versions of the book. The `getBookInfo` method, when called on a Textbook, returns a string that also includes the edition information, as shown in the example.

Information about the book title and price must be maintained in the Book class. Information about the edition must be maintained in the Textbook class.

The Textbook class contains an additional method, `canSubstituteFor`, which returns `true` if a Textbook is a valid substitute for another Textbook and returns `false` otherwise. The current Textbook is a valid substitute for the Textbook referenced by the parameter of the `canSubstituteFor` method if the two Textbook objects have the same title and if the edition of the current Textbook is greater than or equal to the edition of the parameter.

**GO ON TO THE NEXT PAGE.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than Book or Textbook.

Statement	Value Returned (blank if no value)	Class Specification
Textbook bio2015 = new Textbook("Biology", 49.75, 2);		bio2015 is a Textbook with a title of "Biology", a price of 49.75, and an edition of 2.
Textbook bio2019 = new Textbook("Biology", 39.75, 3);		bio2019 is a Textbook with a title of "Biology", a price of 39.75, and an edition of 3.
bio2019.getEdition();	3	The edition is returned.
bio2019.getBookInfo();	"Biology-39.75-3"	The formatted string containing the title, price, and edition of bio2019 is returned.
bio2019. canSubstituteFor(bio2015);	true	bio2019 is a valid substitute for bio2015, since their titles are the same and the edition of bio2019 is greater than or equal to the edition of bio2015.
bio2015. canSubstituteFor(bio2019);	false	bio2015 is not a valid substitute for bio2019, since the edition of bio2015 is less than the edition of bio2019.
Textbook math = new Textbook("Calculus", 45.25, 1);		math is a Textbook with a title of "Calculus", a price of 45.25, and an edition of 1.
bio2015. canSubstituteFor(math);	false	bio2015 is not a valid substitute for math, since the title of bio2015 is not the same as the title of math.

Write the complete Textbook class. Your implementation must meet all specifications and conform to the examples shown in the preceding table.

---

**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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (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.

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (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).

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

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.

"0-Good! Thx."	"2-Great!"	"3-Poor! Bad."
----------------	------------	----------------

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.**

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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

- (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.**

© 2022 College Board.  
Visit College Board on the web: collegeboard.org.

**STOP**

**END OF EXAM**

**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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

- (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");
```

<b>Value of guess</b>	<b>Number of Substring Occurrences</b>	<b>Score Calculation: (Number of Substring Occurrences) x (Square of the Length of guess)</b>	<b>Return Value of game.scoreGuess(guess)</b>
"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");
```

<b>Value of guess</b>	<b>Number of Substring Occurrences</b>	<b>Score Calculation: (Number of Substring Occurrences) x (Square of the Length of guess)</b>	<b>Return Value of game.scoreGuess(guess)</b>
"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)
```

**GO ON TO THE NEXT PAGE.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

- (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
<code>game.scoreGuess("ten");</code>	9	<code>1 * 3 * 3</code>
<code>game.scoreGuess("nation");</code>	36	<code>1 * 6 * 6</code>
<code>game.findBetterGuess("ten", "nation");</code>	"nation"	Since <code>scoreGuess</code> returns 36 for "nation" and 9 for "ten", the guess with the greater score, "nation", is returned.
<code>game.scoreGuess("con");</code>	9	<code>1 * 3 * 3</code>
<code>game.scoreGuess("cat");</code>	9	<code>1 * 3 * 3</code>
<code>game.findBetterGuess("con", "cat");</code>	"con"	Since <code>scoreGuess</code> 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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

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
<code>CombinedTable c1 = new CombinedTable(t1, t2);</code>		A <code>CombinedTable</code> is composed of two <code>SingleTable</code> objects.
<code>c1.canSeat(9);</code>	<code>true</code>	Since its two single tables have a total of 12 seats, <code>c1</code> can seat 10 or fewer people.
<code>c1.canSeat(11);</code>	<code>false</code>	<code>c1</code> cannot seat 11 people.
<code>c1.getDesirability();</code>	<code>65.0</code>	Because <code>c1</code> 's two single tables are the same height, its desirability is the average of 60.0 and 70.0.
<code>CombinedTable c2 = new CombinedTable(t2, t3);</code>		A <code>CombinedTable</code> is composed of two <code>SingleTable</code> objects.
<code>c2.canSeat(18);</code>	<code>true</code>	Since its two single tables have a total of 20 seats, <code>c2</code> can seat 18 or fewer people.
<code>c2.getDesirability();</code>	<code>62.5</code>	Because <code>c2</code> 's two single tables are not the same height, its desirability is 10 units less than the average of 70.0 and 75.0.
<code>t2.setViewQuality(80);</code>		Changing the view quality of one of the tables that makes up <code>c2</code> changes the desirability of <code>c2</code> , as illustrated in the next line of the chart. Since <code>setViewQuality</code> is a <code>SingleTable</code> method, you do not need to write it.
<code>c2.getDesirability();</code>	<code>67.5</code>	Because the view quality of <code>t2</code> changed, the desirability of <code>c2</code> 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.**

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.**

**GO ON TO THE NEXT PAGE.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

(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)
```

**GO ON TO THE NEXT PAGE.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

- (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 <code>isNonZeroRow</code>	Value Returned	Explanation
<code>ArrayResizer.isNonZeroRow(arr, 0)</code>	<code>false</code>	At least one value in row 0 is zero.
<code>ArrayResizer.isNonZeroRow(arr, 1)</code>	<code>true</code>	All values in row 1 are non-zero.
<code>ArrayResizer.isNonZeroRow(arr, 2)</code>	<code>false</code>	At least one value in row 2 is zero.
<code>ArrayResizer.isNonZeroRow(arr, 3)</code>	<code>true</code>	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.

**GO ON TO THE NEXT PAGE.**

- (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.**

© 2021 College Board.  
Visit College Board on the web: collegeboard.org.

**STOP**

**END OF EXAM**

**2019**

**AP®**

 CollegeBoard

---

# **AP® Computer Science A**

## **Free-Response Questions**

**2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 interface and 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.

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

1. The APCalendar class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{
    /** Returns true if year is a leap year and false otherwise. */
    private static boolean isLeapYear(int year)
    { /* implementation not shown */ }

    /** Returns the number of leap years between year1 and year2, inclusive.
     * Precondition: 0 <= year1 <= year2
     */
    public static int numberOfLeapYears(int year1, int year2)
    { /* to be implemented in part (a) */ }

    /** Returns the value representing the day of the week for the first day of year,
     * where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
     */
    private static int firstDayOfYear(int year)
    { /* implementation not shown */ }

    /** Returns n, where month, day, and year specify the nth day of the year.
     * Returns 1 for January 1 (month = 1, day = 1) of any year.
     * Precondition: The date represented by month, day, year is a valid date.
     */
    private static int dayOfYear(int month, int day, int year)
    { /* implementation not shown */ }

    /** Returns the value representing the day of the week for the given date
     * (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
     * and 6 denotes Saturday.
     * Precondition: The date represented by month, day, year is a valid date.
     */
    public static int dayOfWeek(int month, int day, int year)
    { /* to be implemented in part (b) */ }

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

## **2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.  
 * Precondition: 0 <= year1 <= year2  
 */  
public static int numberOfLeapYears(int year1, int year2)
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date (`month`, `day`, `year`), where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns `n`, where `month`, `day`, and `year` specify the `n`th day of the year. For the first day of the year, January 1 (`month = 1`, `day = 1`), the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar  
  
private static boolean isLeapYear(int year)  
public static int numberofLeapYears(int year1, int year2)  
private static int firstDayOfYear(int year)  
private static int dayOfYear(int month, int day, int year)  
public static int dayOfWeek(int month, int day, int year)
```

## **2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 * (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 * and 6 denotes Saturday.
 * Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. This question involves the implementation of a fitness tracking system that is represented by the `StepTracker` class. A `StepTracker` object is created with a parameter that defines the minimum number of steps that must be taken for a day to be considered *active*.

The `StepTracker` class provides a constructor and the following methods.

- `addDailySteps`, which accumulates information about steps, in readings taken once per day
- `activeDays`, which returns the number of active days
- `averageSteps`, which returns the average number of steps per day, calculated by dividing the total number of steps taken by the number of days tracked

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

Statements and Expressions	Value Returned (blank if no value)	Comment
<code>StepTracker tr = new StepTracker(10000);</code>		Days with at least 10,000 steps are considered active. Assume that the parameter is positive.
<code>tr.activeDays();</code>	0	No data have been recorded yet.
<code>tr.averageSteps();</code>	0.0	When no step data have been recorded, the <code>averageSteps</code> method returns 0.0.
<code>tr.addDailySteps(9000);</code>		This is too few steps for the day to be considered active.
<code>tr.addDailySteps(5000);</code>		This is too few steps for the day to be considered active.
<code>tr.activeDays();</code>	0	No day had at least 10,000 steps.
<code>tr.averageSteps();</code>	7000.0	The average number of steps per day is (14000 / 2).
<code>tr.addDailySteps(13000);</code>		This represents an active day.
<code>tr.activeDays();</code>	1	Of the three days for which step data were entered, one day had at least 10,000 steps.
<code>tr.averageSteps();</code>	9000.0	The average number of steps per day is (27000 / 3).
<code>tr.addDailySteps(23000);</code>		This represents an active day.
<code>tr.addDailySteps(1111);</code>		This is too few steps for the day to be considered active.
<code>tr.activeDays();</code>	2	Of the five days for which step data were entered, two days had at least 10,000 steps.
<code>tr.averageSteps();</code>	10222.2	The average number of steps per day is (51111 / 5).

## **2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Write the complete `StepTracker` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.

## **2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.
}
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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)
```

## **2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `getDelimitersList` below.

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

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

(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.

<code>"&lt;sup&gt;"</code>	<code>"&lt;sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>	<code>"&lt;sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>
----------------------------	----------------------------	-----------------------------	----------------------------	-----------------------------	-----------------------------

### Example 2

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

<code>"&lt;sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>	<code>"&lt;sup&gt;"</code>
----------------------------	-----------------------------	-----------------------------	----------------------------



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`.

<code>"&lt;/sup&gt;"</code>
-----------------------------



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.

<code>"&lt;sup&gt;"</code>	<code>"&lt;sup&gt;"</code>	<code>"&lt;/sup&gt;"</code>
----------------------------	----------------------------	-----------------------------

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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)
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.
}
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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)
```

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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 <code>evaluateLight</code>	Value Returned	Explanation
<code>sim.evaluateLight(0, 3);</code>	false	The light is on, and the number of lights that are on in its column is even.
<code>sim.evaluateLight(6, 0);</code>	true	The light is off, and the number of lights that are on in its column is divisible by 3.
<code>sim.evaluateLight(4, 1);</code>	false	Returns the light's current status.
<code>sim.evaluateLight(5, 4);</code>	true	Returns the light's current status.

## 2019 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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**

**2018**

**AP®**

 CollegeBoard

---

# **AP Computer Science A**

## **Free-Response Questions**

**2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 interface and 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.

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following FrogSimulation class. You will write two of the methods in this class.

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;

    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     * position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     * Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
        goalDistance = dist;
        maxHops = numHops;
    }

    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    { /* implementation not shown */ }

    /** Simulates a frog attempting to reach the goal as described in part (a).
     * Returns true if the frog successfully reached or passed the goal during the simulation;
     * false otherwise.
     */
    public boolean simulate()
    { /* to be implemented in part (a) */ }

    /** Runs num simulations and returns the proportion of simulations in which the frog
     * successfully reached or passed the goal.
     * Precondition: num > 0
     */
    public double runSimulations(int num)
    { /* to be implemented in part (b) */ }
}
```

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by <code>hopDistance()</code>	Final position of frog	Return value of <code>sim.simulate()</code>
Example 1	5, 7, -2, 8, 6	24	true
Example 2	6, 7, 6, 6	25	true
Example 3	6, -6, 31	31	true
Example 4	4, 2, -8	-2	false
Example 5	5, 4, 2, 4, 3	18	false

Class information for this question

```
public class FrogSimulation

    private int goalDistance
    private int maxHops

    private int hopDistance()
    public boolean simulate()
    public double runSimulations(int num)
```

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 * Returns true if the frog successfully reached or passed the goal during the simulation;
 * false otherwise.
 */
public boolean simulate()
```

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is 400, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return 0.25.

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

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
```

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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) */ }

}
```

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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  $0 \leq i < j < \text{words.length}$ . 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()
```

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete the WordPairList constructor below.

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

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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()
```

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `numMatches` below.

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

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. The `StringChecker` interface describes classes that check if strings are valid, according to some criterion.

```
public interface StringChecker
{
    /** Returns true if str is valid. */
    boolean isValid(String str);
}
```

A `CodeWordChecker` is a `StringChecker`. A `CodeWordChecker` object can be constructed with three parameters: two integers and a string. The first two parameters specify the minimum and maximum code word lengths, respectively, and the third parameter specifies a string that must not occur in the code word. A `CodeWordChecker` object can also be constructed with a single parameter that specifies a string that must not occur in the code word; in this case the minimum and maximum lengths will default to 6 and 20, respectively.

The following examples illustrate the behavior of `CodeWordChecker` objects.

### Example 1

```
StringChecker sc1 = new CodeWordChecker(5, 8, "$");
```

Valid code words have 5 to 8 characters and must not include the string "\$".

Method call	Return value	Explanation
<code>sc1.isValid("happy")</code>	true	The code word is valid.
<code>sc1.isValid("happy\$")</code>	false	The code word contains "\$".
<code>sc1.isValid("Code")</code>	false	The code word is too short.
<code>sc1.isValid("happyCode")</code>	false	The code word is too long.

### Example 2

```
StringChecker sc2 = new CodeWordChecker("pass");
```

Valid code words must not include the string "pass". Because the bounds are not specified, the length bounds are 6 and 20, inclusive.

Method call	Return value	Explanation
<code>sc2.isValid("MyPass")</code>	true	The code word is valid.
<code>sc2.isValid("Mypassport")</code>	false	The code word contains "pass".
<code>sc2.isValid("happy")</code>	false	The code word is too short.
<code>sc2.isValid("1,000,000,000,000,000")</code>	false	The code word is too long.

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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) */ }
}
```

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `getColumn` below.

```
/** 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)
```

## 2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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

1	2	3
3	1	2
7	8	9

1	2
1	2

Not a Latin square  
because the first row  
contains duplicate  
values

Not a Latin square  
because the elements of  
the first row do not all  
appear in the third row

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)
```

## **2018 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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**

2017



---

# AP Computer Science A

## Free-Response Questions



**2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 interface and 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.

## 2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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) */ }
}
```

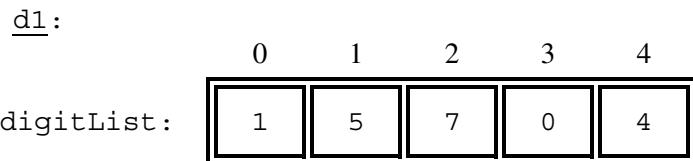
Part (a) begins on page 4.

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

© 2017 The College Board.  
Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

**GO ON TO THE NEXT PAGE.**

**2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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
<code>new Digits(7).isStrictlyIncreasing()</code>	<code>true</code>
<code>new Digits(1356).isStrictlyIncreasing()</code>	<code>true</code>
<code>new Digits(1336).isStrictlyIncreasing()</code>	<code>false</code>
<code>new Digits(1536).isStrictlyIncreasing()</code>	<code>false</code>
<code>new Digits(65310).isStrictlyIncreasing()</code>	<code>false</code>

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `isStrictlyIncreasing` below.

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

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

2. This question involves the design of a class that will be used to produce practice problems. The following `StudyPractice` interface represents practice problems that can be used to study some subject.

```
public interface StudyPractice
{
    /** Returns the current practice problem. */
    String getProblem();

    /** Changes to the next practice problem. */
    void nextProblem();
}
```

The `MultPractice` class is a `StudyPractice` that produces multiplication practice problems. A `MultPractice` object is constructed with two integer values: *first integer* and *initial second integer*. The first integer is a value that remains constant and is used as the first integer in every practice problem. The initial second integer is used as the starting value for the second integer in the practice problems. This second value is incremented for each additional practice problem that is produced by the class.

For example, a `MultPractice` object created with the call `new MultPractice(7, 3)` would be used to create the practice problems "7 TIMES 3", "7 TIMES 4", "7 TIMES 5", and so on.

In the `MultPractice` class, the `getProblem` method returns a string in the format of "*first integer* TIMES *second integer*". The `nextProblem` method updates the state of the `MultPractice` object to represent the next practice problem.

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

The following examples illustrate the behavior of the `MultPractice` class. Each table shows a code segment and the output that would be produced as the code is executed.

### **Example 1**

Code segment	Output produced
<pre>StudyPractice p1 = new MultPractice(7, 3); System.out.println(p1.getProblem());</pre>	7 TIMES 3
<pre>p1.nextProblem(); System.out.println(p1.getProblem());</pre>	7 TIMES 4
<pre>p1.nextProblem(); System.out.println(p1.getProblem());</pre>	7 TIMES 5
<pre>p1.nextProblem(); System.out.println(p1.getProblem());</pre>	7 TIMES 6

### **Example 2**

Code segment	Output produced
<pre>StudyPractice p2 = new MultPractice(4, 12); p2.nextProblem(); System.out.println(p2.getProblem()); System.out.println(p2.getProblem());</pre>	4 TIMES 13 4 TIMES 13
<pre>p2.nextProblem(); p2.nextProblem(); System.out.println(p2.getProblem());</pre>	4 TIMES 15
<pre>p2.nextProblem(); System.out.println(p2.getProblem());</pre>	4 TIMES 16

Question 2 continues on page 10.

**2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Interface information for this question

public interface StudyPractice

String getProblem()  
void nextProblem()

Write the complete `MultPractice` class. Your implementation must be consistent with the specifications and the given examples.

## 2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. This question involves analyzing and modifying a string. The following `Phrase` class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the `Phrase` class.

```
public class Phrase
{
    private String currentPhrase;

    /** Constructs a new Phrase object. */
    public Phrase(String p)
    {   currentPhrase = p;   }

    /** Returns the index of the nth occurrence of str in the current phrase;
     *  returns -1 if the nth occurrence does not exist.
     *  Precondition: str.length() > 0 and n > 0
     *  Postcondition: the current phrase is not modified.
     */
    public int findNthOccurrence(String str, int n)
    {   /* implementation not shown */   }

    /** Modifies the current phrase by replacing the nth occurrence of str with repl.
     *  If the nth occurrence does not exist, the current phrase is unchanged.
     *  Precondition: str.length() > 0 and n > 0
     */
    public void replaceNthOccurrence(String str, int n, String repl)
    {   /* to be implemented in part (a) */   }

    /** Returns the index of the last occurrence of str in the current phrase;
     *  returns -1 if str is not found.
     *  Precondition: str.length() > 0
     *  Postcondition: the current phrase is not modified.
     */
    public int findLastOccurrence(String str)
    {   /* to be implemented in part (b) */   }

    /** Returns a string containing the current phrase. */
    public String toString()
    {   return currentPhrase;   }
}
```

Part (a) begins on page 12.

## 2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the `Phrase` method `replaceNthOccurrence`, which will replace the `n`th occurrence of the string `str` with the string `repl`. If the `n`th occurrence does not exist, `currentPhrase` remains unchanged.

Several examples of the behavior of the method `replaceNthOccurrence` are shown below.

Code segments

Output produced

Phrase phrase1 = new Phrase("A cat ate late."); phrase1.replaceNthOccurrence("at", 1, "rane"); System.out.println(phrase1);	A crane ate late.
---	-------------------

Phrase phrase2 = new Phrase("A cat ate late."); phrase2.replaceNthOccurrence("at", 6, "xx"); System.out.println(phrase2);	A cat ate late.
---	-----------------

Phrase phrase3 = new Phrase("A cat ate late."); phrase3.replaceNthOccurrence("bat", 2, "xx"); System.out.println(phrase3);	A cat ate late.
--	-----------------

Phrase phrase4 = new Phrase("aaaa"); phrase4.replaceNthOccurrence("aa", 1, "xx"); System.out.println(phrase4);	xxaa
--	------

Phrase phrase5 = new Phrase("aaaa"); phrase5.replaceNthOccurrence("aa", 2, "bbb"); System.out.println(phrase5);	abbba
---	-------

Class information for this question

```
public class Phrase  
  
    private String currentPhrase  
    public Phrase(String p)  
    public int findNthOccurrence(String str, int n)  
    public void replaceNthOccurrence(String str, int n, String repl)  
    public int findLastOccurrence(String str)  
    public String toString()
```

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

The `Phrase` class includes the method `findNthOccurrence`, which returns the `n`th occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.  
 * If the nth occurrence does not exist, the current phrase is unchanged.  
 * Precondition: str.length() > 0 and n > 0  
 */  
public void replaceNthOccurrence(String str, int n, String repl)
```

Part (b) begins on page 14.

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) Write the `Phrase` method `findLastOccurrence`. This method finds and returns the index of the last occurrence of a given string in `currentPhrase`. If the given string is not found, `-1` is returned. The following tables show several examples of the behavior of the method `findLastOccurrence`.

```
Phrase phrase1 = new Phrase("A cat ate late.");
```

Method call	Value returned
<code>phrase1.findLastOccurrence("at")</code>	11
<code>phrase1.findLastOccurrence("cat")</code>	2
<code>phrase1.findLastOccurrence("bat")</code>	-1

Class information for this question

```
public class Phrase

private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;  
 * returns -1 if str is not found.  
 * Precondition: str.length() > 0  
 * Postcondition: the current phrase is not modified.  
 */  
public int findLastOccurrence(String str)
```

## 2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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`.

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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				2D Successor Array					
	0	1	2	3	0	1	2	3	
0	15	5	9	10	0	$(1, 1)$	$(1, 3)$	$(0, 3)$	$(1, 2)$
1	12	16	11	6	1	$(2, 2)$	<code>null</code>	$(1, 0)$	$(2, 3)$
2	14	8	13	7	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)
```

## **2017 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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**



---

# **AP<sup>®</sup> Computer Science A**

## **2016 Free-Response Questions**

---

© 2016 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).

**2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.
}
```

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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
```

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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*.

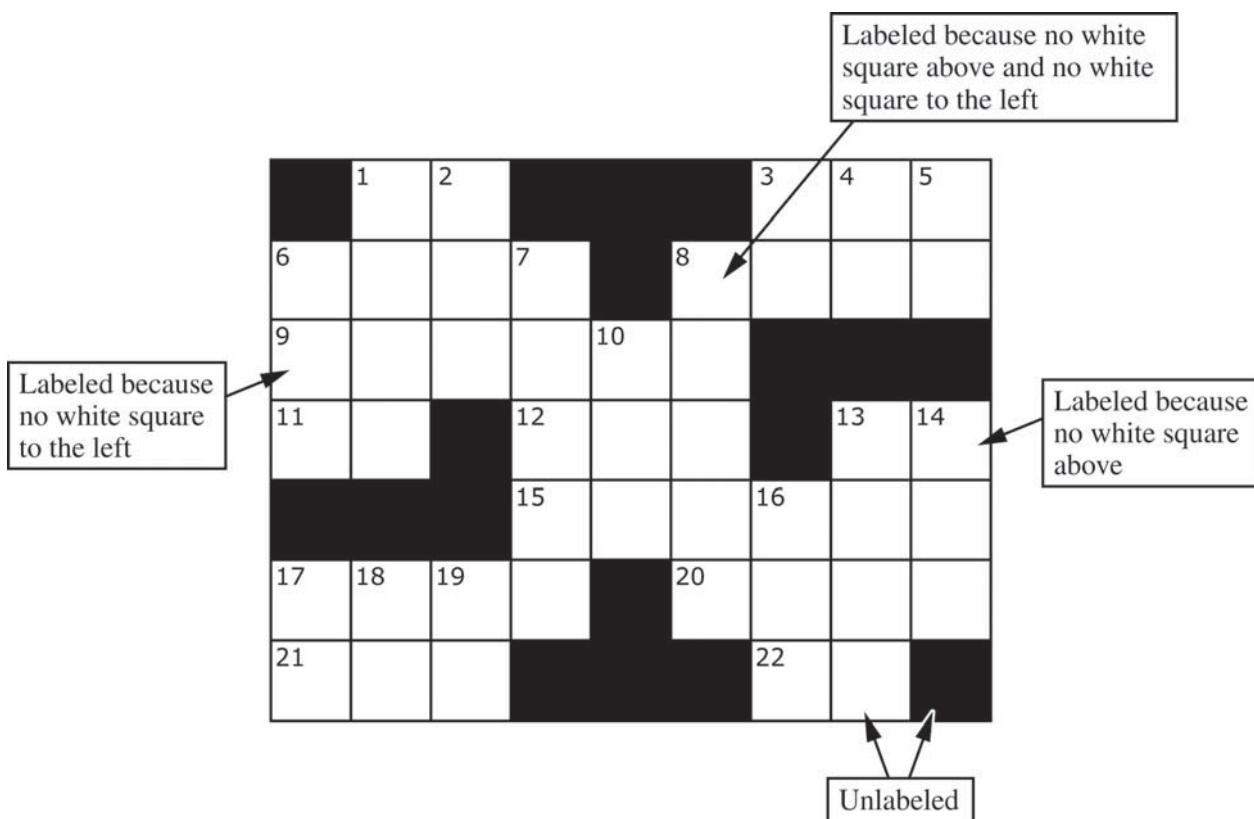
The crossword labeling rule identifies squares to be labeled with a positive number as follows.

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.



## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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.**

## **2016 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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**



---

# **AP<sup>®</sup> Computer Science A**

## **2015 Free-Response Questions**

---

© 2015 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).

**2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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`.

<u>arr1</u>					Value returned by <u>arraySum(arr1)</u>
0	1	2	3	4	
1	3	2	7	3	16

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## **2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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}`.

		<u>mat1</u>				
		0	1	2	3	4
0	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.**

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## 2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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.

<u>mat1</u>					Row sums
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

<u>mat2</u>					Row sums
0	1	2	3	4	
0	1	1	5	3	4
1	12	7	6	1	9
2	8	11	10	2	5
3	3	2	3	0	6

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)
```

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## **2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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

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

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

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

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

```
HiddenWord puzzle = new HiddenWord("HARPS");
```

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

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

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

## 2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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; }
}
```

## 2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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:  $0 \leq \text{row} < \text{getNumRows}()$ 
     *  $0 \leq \text{col} < \text{getNumCols}()$ 
     */
    public int getValueAt(int row, int col)
    {   /* to be implemented in part (a) */   }

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

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

## 2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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`

<code>entries:</code>	<table border="1"><tr><td><code>row: 1</code></td><td><code>row: 2</code></td><td><code>row: 3</code></td><td><code>row: 1</code></td></tr><tr><td><code>col: 4</code></td><td><code>col: 0</code></td><td><code>col: 1</code></td><td><code>col: 1</code></td></tr><tr><td><code>value: 4</code></td><td><code>value: 1</code></td><td><code>value: -9</code></td><td><code>value: 5</code></td></tr></table>	<code>row: 1</code>	<code>row: 2</code>	<code>row: 3</code>	<code>row: 1</code>	<code>col: 4</code>	<code>col: 0</code>	<code>col: 1</code>	<code>col: 1</code>	<code>value: 4</code>	<code>value: 1</code>	<code>value: -9</code>	<code>value: 5</code>
<code>row: 1</code>	<code>row: 2</code>	<code>row: 3</code>	<code>row: 1</code>										
<code>col: 4</code>	<code>col: 0</code>	<code>col: 1</code>	<code>col: 1</code>										
<code>value: 4</code>	<code>value: 1</code>	<code>value: -9</code>	<code>value: 5</code>										

- 
- (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.

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `getValueAt` below.

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

Part (b) begins on page 13.

## 2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

(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`

<code>entries:</code>	<table border="1"><tr><td><code>row: 1</code></td><td><code>row: 2</code></td><td><code>row: 3</code></td><td><code>row: 1</code></td></tr><tr><td><code>col: 4</code></td><td><code>col: 0</code></td><td><code>col: 1</code></td><td><code>col: 1</code></td></tr><tr><td><code>value: 4</code></td><td><code>value: 1</code></td><td><code>value: -9</code></td><td><code>value: 5</code></td></tr></table>	<code>row: 1</code>	<code>row: 2</code>	<code>row: 3</code>	<code>row: 1</code>	<code>col: 4</code>	<code>col: 0</code>	<code>col: 1</code>	<code>col: 1</code>	<code>value: 4</code>	<code>value: 1</code>	<code>value: -9</code>	<code>value: 5</code>
<code>row: 1</code>	<code>row: 2</code>	<code>row: 3</code>	<code>row: 1</code>										
<code>col: 4</code>	<code>col: 0</code>	<code>col: 1</code>	<code>col: 1</code>										
<code>value: 4</code>	<code>value: 1</code>	<code>value: -9</code>	<code>value: 5</code>										

---

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`

<code>entries:</code>	<table border="1"><tr><td><code>row: 1</code></td><td><code>row: 2</code></td></tr><tr><td><code>col: 3</code></td><td><code>col: 0</code></td></tr><tr><td><code>value: 4</code></td><td><code>value: 1</code></td></tr></table>	<code>row: 1</code>	<code>row: 2</code>	<code>col: 3</code>	<code>col: 0</code>	<code>value: 4</code>	<code>value: 1</code>
<code>row: 1</code>	<code>row: 2</code>						
<code>col: 3</code>	<code>col: 0</code>						
<code>value: 4</code>	<code>value: 1</code>						

## **2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `removeColumn` below.

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

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.

## **2015 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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
<code>multiple1.contains(2)</code>	<code>true</code>
<code>multiple1.contains(9)</code>	<code>false</code>
<code>multiple1.contains(6)</code>	<code>true</code>

## **2015 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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<sup>®</sup> Computer Science A**

## **2014 Free-Response Questions**

---

© 2014 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).

**2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 <code>scrambleWord(word)</code>
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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.

## **2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (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.**

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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)
```

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.
}
```

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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.

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

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

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

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

Part (a) continues on page 9.

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

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.

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (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)

```

## **2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

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)
```

## 2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

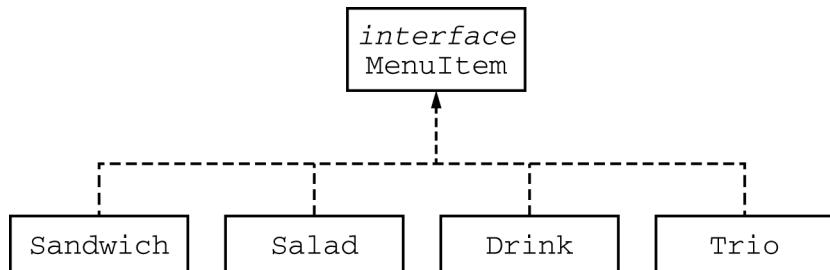
4. The menu at a lunch counter includes a variety of sandwiches, salads, and drinks. The menu also allows a customer to create a "trio," which consists of three menu items: a sandwich, a salad, and a drink. The price of the trio is the sum of the two highest-priced menu items in the trio; one item with the lowest price is free.

Each menu item has a name and a price. The four types of menu items are represented by the four classes Sandwich, Salad, Drink, and Trio. All four classes implement the following MenuItem interface.

```
public interface MenuItem
{
    /** @return the name of the menu item */
    String getName();

    /** @return the price of the menu item */
    double getPrice();
}
```

The following diagram shows the relationship between the MenuItem interface and the Sandwich, Salad, Drink, and Trio classes.



For example, assume that the menu includes the following items. The objects listed under each heading are instances of the class indicated by the heading.

Sandwich	Salad	Drink
"Cheeseburger" 2.75	"Spinach Salad" 1.25	"Orange Soda" 1.25
"Club Sandwich" 2.75	"Coleslaw" 1.25	"Cappuccino" 3.50

Question 4 continues on page 13

## **2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

The menu allows customers to create `Trio` menu items, each of which includes a sandwich, a salad, and a drink. The name of the `Trio` consists of the names of the sandwich, salad, and drink, in that order, each separated by `"/ "` and followed by a space and then `"Trio"`. The price of the `Trio` is the sum of the two highest-priced items in the `Trio`; one item with the lowest price is free.

A trio consisting of a cheeseburger, spinach salad, and an orange soda would have the name `"Cheeseburger/Spinach Salad/Orange Soda Trio"` and a price of \$4.00 (the two highest prices are \$2.75 and \$1.25). Similarly, a trio consisting of a club sandwich, coleslaw, and a cappuccino would have the name `"Club Sandwich/Coleslaw/Cappuccino Trio"` and a price of \$6.25 (the two highest prices are \$2.75 and \$3.50).

Write the `Trio` class that implements the `MenuItem` interface. Your implementation must include a constructor that takes three parameters representing a sandwich, salad, and drink. The following code segment should have the indicated behavior.

```
Sandwich sandwich;
Salad salad;
Drink drink;
/* Code that initializes sandwich, salad, and drink */

Trio trio = new Trio(sandwich, salad, drink); // Compiles without error

Trio trio1 = new Trio(salad, sandwich, drink); // Compile-time error
Trio trio2 = new Trio(sandwich, salad, salad); // Compile-time error
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

**2014 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Write the complete `Trio` class below.

**STOP**

**END OF EXAM**



---

# **AP<sup>®</sup> Computer Science A**

## **2014 Scoring Guidelines**

---

© 2014 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).

# AP® COMPUTER SCIENCE A

## 2014 GENERAL SCORING GUIDELINES

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question.

### **1-Point Penalty**

- (w) Extraneous code that causes side effect (*e.g., writing to output, failure to compile*)
- (x) Local variables used but none declared
- (y) Destruction of persistent data (*e.g., changing value referenced by parameter*)
- (z) Void method or constructor that returns a value

### **No Penalty**

- Extraneous code with no side effect (*e.g., precondition check, no-op*)
- Spelling/case discrepancies where there is no ambiguity\*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators ( $\times \bullet \div \leq \geq < > \neq$ )
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- Array/collection access confusion (`[] get`)
- `length`/`size` confusion for array, `String`, `List`, or `ArrayList`, with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i, j]` instead of `[i] [j]`
- Extraneous size in array declaration, *e.g.*, `int[size] nums = new int[size];`
- Missing `;` provided majority are present and indentation clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

\**Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be **unambiguously** inferred from context; for example, “ArayList” instead of “ArrayList”. As a counterexample, note that if the code declares “Bug bug;”, then uses “Bug.move ()” instead of “bug.move ()”, the context does **not** allow for the reader to assume the object instead of the class.*

# AP® COMPUTER SCIENCE A 2014 SCORING GUIDELINES

## Question 1: Word Scramble

Part (a)	scrambleWord	5 points
----------	--------------	----------

**Intent:** Scramble a word by swapping all letter pairs that begin with A

- +1 Accesses all letters in word, left to right (*no bounds errors*)
- +1 Identifies at least one letter pair consisting of “A” followed by non-“A”
- +1 Reverses identified pair in constructing result string
- +1 Constructs correct result string (*Point lost if any letters swapped more than once, minor loop bounds errors ok*)
- +1 Returns constructed string

Part (b)	scrambleOrRemove	4 points
----------	------------------	----------

**Intent:** Modify list by replacing each word with scrambled version and removing any word unchanged by scrambling

- +1 Accesses all words in wordList (*no bounds errors*)
- +1 Calls scrambleWord with a word from the list as parameter
- +1 Identifies words unchanged by scrambling
- +1 On exit: List includes all and only words that have been changed by scrambling once, in their original relative order (*minor loop bounds errors ok*)

# AP® COMPUTER SCIENCE A 2014 SCORING GUIDELINES

## Question 3: Seating Chart

Part (a)	SeatingChart constructor	5 points
----------	--------------------------	----------

**Intent:** Create SeatingChart object from list of students

- +1 seats = new Student [rows] [cols]; (or equivalent code)
- +1 Accesses all elements of studentList (no bounds errors on studentList)
- +1 Accesses all necessary elements of seats array (no bounds errors on seats array, point lost if access not column-major order)
- +1 Assigns value from studentList to at least one element in seats array
- +1 On exit: All elements of seats have correct values (minor loop bounds errors ok)

Part (b)	removeAbsentStudents	5 points
----------	----------------------	----------

**Intent:** Remove students with more than given number of absences from seating chart and return count of students removed

- +1 Accesses all elements of seats (no bounds errors)
- +1 Calls getAbsenceCount() on Student object (point lost if null case not handled correctly)
- +1 Assigns null to all elements in seats array when absence count for occupying student > allowedAbsences (point lost if seats array element changed in other cases)
- +1 Computes and returns correct number of students removed

Question-Specific Penalties
-----------------------------

- 2 (v) Consistently uses incorrect array name instead of seats or studentList

# AP® COMPUTER SCIENCE A 2014 SCORING GUIDELINES

## Question 4: Trio

<b>Class:</b>	Trio	<b>9 points</b>
---------------	------	-----------------

**Intent:** Define implementation of MenuItem interface that consists of sandwich, salad, and drink

- +1 public class Trio implements MenuItem
- +1 Declares appropriate private instance variables
- +2 Implements constructor
  - +1 public Trio(Sandwich sandwich, Salad salad, Drink drink)
  - +1 Initializes appropriate instance variables using parameters
- +1 Implements interface methods  
(public String getName () {...}, public double getPrice () {...})
- +1 Constructs correct name string and makes available for return in getName
- +1 Returns constructed name string in getName
- +1 Computes correct price and makes available for return in getPrice
- +1 Returns computed price in getPrice

### Question-Specific Penalties

- 0 Missing or extra spaces in name string, “trio”
- 1 (w) Extraneous default constructor that causes side effect



## **AP<sup>®</sup> Computer Science A 2013 Free-Response Questions**

### **About the College Board**

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of more than 6,000 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT<sup>®</sup> and the Advanced Placement Program<sup>®</sup>. The organization also serves the education community through research and advocacy on behalf of students, educators, and schools.

© 2013 The College Board. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service and inspiring minds are trademarks owned by the College Board. All other products and services may be trademarks of their respective owners. Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org). Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.org/inquiry/cbpermit.html](http://www.collegeboard.org/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).



**2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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. A music Web site keeps track of downloaded music. For each download, the site uses a `DownloadInfo` object to store a song's title and the number of times it has been downloaded. A partial declaration for the `DownloadInfo` class is shown below.

```
public class DownloadInfo
{
    /** Creates a new instance with the given unique title and sets the
     *  number of times downloaded to 1.
     *  @param title the unique title of the downloaded song
     */
    public DownloadInfo(String title)
    { /* implementation not shown */ }

    /** @return the title */
    public String getTitle()
    { /* implementation not shown */ }

    /** Increment the number times downloaded by 1 */
    public void incrementTimesDownloaded()
    { /* implementation not shown */ }

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

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The list of downloaded information is stored in a `MusicDownloads` object. A partial declaration for the `MusicDownloads` class is shown below.

```
public class MusicDownloads
{
    /** The list of downloaded information.
     * Guaranteed not to be null and not to contain duplicate titles.
     */
    private List<DownloadInfo> downloadList;

    /** Creates the list of downloaded information. */
    public MusicDownloads()
    {   downloadList = new ArrayList<DownloadInfo>();   }

    /** Returns a reference to the DownloadInfo object with the requested title if it exists.
     * @param title the requested title
     * @return a reference to the DownloadInfo object with the
     *         title that matches the parameter title if it exists in the list;
     *         null otherwise.
     * Postcondition:
     * - no changes were made to downloadList.
     */
    public DownloadInfo getDownloadInfo(String title)
    {   /* to be implemented in part (a) */   }

    /** Updates downloadList with information from titles.
     * @param titles a list of song titles
     * Postcondition:
     * - there are no duplicate titles in downloadList.
     * - no entries were removed from downloadList.
     * - all songs in titles are represented in downloadList.
     * - for each existing entry in downloadList, the download count is increased by
     *     the number of times its title appeared in titles.
     * - the order of the existing entries in downloadList is not changed.
     * - the first time an object with a title from titles is added to downloadList, it
     *     is added to the end of the list.
     * - new entries in downloadList appear in the same order
     *     in which they first appear in titles.
     * - for each new entry in downloadList, the download count is equal to
     *     the number of times its title appeared in titles.
     */
    public void updateDownloads(List<String> titles)
    {   /* to be implemented in part (b) */   }

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

Part (a) begins on page 4.

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the `MusicDownloads` method `getDownloadInfo`, which returns a reference to a `DownloadInfo` object if an object with a title that matches the parameter `title` exists in the `downloadList`. If no song in `downloadList` has a title that matches the parameter `title`, the method returns `null`.

For example, suppose variable `webMusicA` refers to an instance of `MusicDownloads` and that the table below represents the contents of `downloadList`. The list contains three `DownloadInfo` objects. The object at position 0 has a title of "Hey Jude" and a download count of 5. The object at position 1 has a title of "Soul Sister" and a download count of 3. The object at position 2 has a title of "Aqualung" and a download count of 10.

0	1	2
"Hey Jude" 5	"Soul Sister" 3	"Aqualung" 10

The call `webMusicA.getDownloadInfo("Aqualung")` returns a reference to the object in position 2 of the list.

The call `webMusicA.getDownloadInfo("Happy Birthday")` returns `null` because there are no `DownloadInfo` objects with that title in the list.

Class information repeated from the beginning of the question

```
public class DownloadInfo  
  
    public DownloadInfo(String title)  
    public String getTitle()  
    public void incrementTimesDownloaded()  
  
public class MusicDownloads  
  
    private List<DownloadInfo> downloadList  
    public DownloadInfo getDownloadInfo(String title)  
    public void updateDownloads(List<String> titles)
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `getDownloadInfo` below.

```
/** Returns a reference to the DownloadInfo object with the requested title if it exists.  
 * @param title the requested title  
 * @return a reference to the DownloadInfo object with the  
 *         title that matches the parameter title if it exists in the list;  
 *         null otherwise.  
 * Postcondition:  
 * - no changes were made to downloadList.  
 */  
public DownloadInfo getDownloadInfo(String title)
```

Part (b) begins on page 6.

© 2013 The College Board.  
Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

**GO ON TO THE NEXT PAGE.**

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the `MusicDownloads` method `updateDownloads`, which takes a list of song titles as a parameter. For each title in the list, the method updates `downloadList`, either by incrementing the download count if a `DownloadInfo` object with the same title exists, or by adding a new `DownloadInfo` object with that title and a download count of 1 to the end of the list. When a new `DownloadInfo` object is added to the end of the list, the order of the already existing entries in `downloadList` remains unchanged.

For example, suppose variable `webMusicB` refers to an instance of `MusicDownloads` and that the table below represents the contents of the instance variable `downloadList`.

0	1	2
"Hey Jude" 5	"Soul Sister" 3	"Aqualung" 10

Assume that the variable `List<String> songTitles` has been defined and contains the following entries.

```
{ "Lights", "Aqualung", "Soul Sister", "Go Now", "Lights", "Soul Sister" }
```

The call `webMusicB.updateDownloads(songTitles)` results in the following `downloadList` with incremented download counts for the objects with titles of "Soul Sister" and "Aqualung". It also has a new `DownloadInfo` object with a title of "Lights" and a download count of 2, and another `DownloadInfo` object with a title of "Go Now" and a download count of 1. The order of the already existing entries remains unchanged.

0	1	2	3	4
"Hey Jude" 5	"Soul Sister" 5	"Aqualung" 11	"Lights" 2	"Go Now" 1

Class information repeated from the beginning of the question

```
public class DownloadInfo  
  
    public DownloadInfo(String title)  
    public String getTitle()  
    public void incrementTimesDownloaded()  
  
public class MusicDownloads  
  
    private List<DownloadInfo> downloadList  
    public DownloadInfo getDownloadInfo(String title)  
    public void updateDownloads(List<String> titles)
```

In writing your solution, you must use the `getDownloadInfo` method. Assume that `getDownloadInfo` works as specified, regardless of what you wrote for part (a).

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `updateDownloads` below.

```
/** Updates downloadList with information from titles.  
 * @param titles a list of song titles  
 * Postcondition:  
 *   - there are no duplicate titles in downloadList.  
 *   - no entries were removed from downloadList.  
 *   - all songs in titles are represented in downloadList.  
 *   - for each existing entry in downloadList, the download count is increased by  
 *     the number of times its title appeared in titles.  
 *   - the order of the existing entries in downloadList is not changed.  
 *   - the first time an object with a title from titles is added to downloadList, it  
 *     is added to the end of the list.  
 *   - new entries in downloadList appear in the same order  
 *     in which they first appear in titles.  
 *   - for each new entry in downloadList, the download count is equal to  
 *     the number of times its title appeared in titles.  
 */  
public void updateDownloads(List<String> titles )
```

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. A multiplayer game called Token Pass has the following rules.

Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered, starting with 0.

The following rules apply for a player's turn.

- The tokens are collected and removed from the game board at that player's position.
- The collected tokens are distributed one at a time, to each player, beginning with the next player in order of increasing position.
- If there are still tokens to distribute after the player at the highest position gets a token, the next token will be distributed to the player at position 0.
- The distribution of tokens continues until there are no more tokens to distribute.

The Token Pass game board is represented by an array of integers. The indexes of the array represent the player positions on the game board, and the corresponding values in the array represent the number of tokens that each player has. The following example illustrates one player's turn.

### Example

The following represents a game with 4 players. The player at position 2 was chosen to go first.

Player	0	1	2	3
Tokens	3	2	6	10

The tokens at position 2 are collected and distributed as follows.

- 1st token - to position 3 (The highest position is reached, so the next token goes to position 0.)
- 2nd token - to position 0
- 3rd token - to position 1
- 4th token - to position 2
- 5th token - to position 3 (The highest position is reached, so the next token goes to position 0.)
- 6th token - to position 0

After player 2's turn, the values in the array will be as follows.

Player	0	1	2	3
Tokens	5	3	1	12

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The Token Pass game is represented by the `TokenPass` class.

```
public class TokenPass
{
    private int[] board;
    private int currentPlayer;

    /**
     * Creates the board array to be of size playerCount and fills it with
     * random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
     * random integer value in the range between 0 and playerCount-1, inclusive.
     * @param playerCount the number of players
     */
    public TokenPass(int playerCount)
    {   /* to be implemented in part (a) */ }

    /**
     * Distributes the tokens from the current player's position one at a time to each player in
     * the game. Distribution begins with the next position and continues until all the tokens
     * have been distributed. If there are still tokens to distribute when the player at the
     * highest position is reached, the next token will be distributed to the player at position 0.
     * Precondition: the current player has at least one token.
     * Postcondition: the current player has not changed.
     */
    public void distributeCurrentPlayerTokens()
    {   /* to be implemented in part (b) */ }

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

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the constructor for the `TokenPass` class. The parameter `playerCount` represents the number of players in the game. The constructor should create the `board` array to contain `playerCount` elements and fill the array with random numbers between 1 and 10, inclusive. The constructor should also initialize the instance variable `currentPlayer` to a random number between 0 and `playerCount-1`, inclusive.

Complete the `TokenPass` constructor below.

```
/** Creates the board array to be of size playerCount and fills it with
 * random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
 * random integer value in the range between 0 and playerCount-1, inclusive.
 * @param playerCount the number of players
 */
public TokenPass(int playerCount)
```

Part (b) begins on page 11.

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) Write the `distributeCurrentPlayerTokens` method.

The tokens are collected and removed from the game board at the current player's position. These tokens are distributed, one at a time, to each player, beginning with the next higher position, until there are no more tokens to distribute.

Class information repeated from the beginning of the question

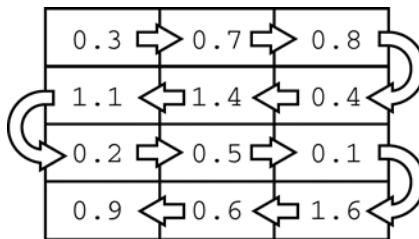
```
public class TokenPass  
  
private int[] board  
private int currentPlayer  
public TokenPass(int playerCount)  
public void distributeCurrentPlayerTokens()
```

Complete method `distributeCurrentPlayerTokens` below.

```
/** Distributes the tokens from the current player's position one at a time to each player in  
 * the game. Distribution begins with the next position and continues until all the tokens  
 * have been distributed. If there are still tokens to distribute when the player at the  
 * highest position is reached, the next token will be distributed to the player at position 0.  
 * Precondition: the current player has at least one token.  
 * Postcondition: the current player has not changed.  
 */  
public void distributeCurrentPlayerTokens()
```

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. A telescope scans a rectangular area of the night sky and collects the data into a 1-dimensional array. Each data value scanned is a number representing the amount of light detected by the telescope. The telescope scans back and forth across the sky (alternating between left to right and right to left) in the pattern indicated below by the arrows. The back-and-forth ordering of the values received from the scan is called *telescope order*.



The telescope records the data in telescope order into a 1-dimensional array of `double` values. This 1-dimensional array of information received from a single scan will be transferred into a 2-dimensional array, which reconstructs the original view of the rectangular area of the sky. This 2-dimensional array is part of the `SkyView` class, shown below. In this question you will write a constructor and a method for this class.

```

public class SkyView
{
    /** A rectangular array that holds the data representing a rectangular area of the sky. */
    private double[][] view;

    /** Constructs a SkyView object from a 1-dimensional array of scan data.
     * @param numRows the number of rows represented in the view
     * Precondition: numRows > 0
     * @param numCols the number of columns represented in the view
     * Precondition: numCols > 0
     * @param scanned the scan data received from the telescope, stored in telescope order
     * Precondition: scanned.length == numRows * numCols
     * Postcondition: view has been created as a rectangular 2-dimensional array
     * with numRows rows and numCols columns and the values in
     * scanned have been copied to view and are ordered as
     * in the original rectangular area of sky.
     */
    public SkyView(int numRows, int numCols, double[] scanned)
    { /* to be implemented in part (a) */ }

    /** Returns the average of the values in a rectangular section of view.
     * @param startRow the first row index of the section
     * @param endRow the last row index of the section
     * @param startCol the first column index of the section
     * @param endCol the last column index of the section
     * Precondition: 0 <= startRow <= endRow < view.length
     * Precondition: 0 <= startCol <= endCol < view[0].length
     * @return the average of the values in the specified section of view
     */
    public double getAverage(int startRow, int endRow,
                           int startCol, int endCol)
    { /* to be implemented in part (b) */ }

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

```

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the constructor for the `SkyView` class. The constructor initializes the `view` instance variable to a 2-dimensional array with `numRows` rows and `numCols` columns. The information from `scanned`, which is stored in the telescope order, is copied into `view` to reconstruct the sky view as originally seen by the telescope. The information in `scanned` must be rearranged as it is stored into `view` so that the sky view is oriented properly.

For example, suppose `scanned` contains values, as shown in the following array.

	0	1	2	3	4	5	6	7	8	9	10	11
scanned	0.3	0.7	0.8	0.4	1.4	1.1	0.2	0.5	0.1	1.6	0.6	0.9

Using the `scanned` array above, a `SkyView` object created with `new SkyView(4, 3, scanned)`, would have `view` initialized with the following values.

	view	0	1	2
0	0.3	0.7	0.8	
1	1.1	1.4	0.4	
2	0.2	0.5	0.1	
3	0.9	0.6	1.6	

For another example, suppose `scanned` contains the following values.

	0	1	2	3	4	5
scanned	0.3	0.7	0.8	0.4	1.4	1.1

A `SkyView` object created with `new SkyView(3, 2, scanned)`, would have `view` initialized with the following values.

	view	0	1
0	0.3	0.7	
1	0.4	0.8	
2	1.4	1.1	

## 2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete the SkyView constructor below.

```
/** Constructs a SkyView object from a 1-dimensional array of scan data.  
 * @param numRows the number of rows represented in the view  
 * Precondition: numRows > 0  
 * @param numCols the number of columns represented in the view  
 * Precondition: numCols > 0  
 * @param scanned the scan data received from the telescope, stored in telescope order  
 * Precondition: scanned.length == numRows * numCols  
 * Postcondition: view has been created as a rectangular 2-dimensional array  
 * with numRows rows and numCols columns and the values in  
 * scanned have been copied to view and are ordered as  
 * in the original rectangular area of sky.  
 */  
public SkyView(int numRows, int numCols, double[] scanned)
```

Part (b) begins on page 19.

© 2013 The College Board.  
Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

**GO ON TO THE NEXT PAGE.**

## **2013 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) Write the `SkyView` method `getAverage`, which returns the average of the elements of the section of `view` with row indexes from `startRow` through `endRow`, inclusive, and column indexes from `startCol` through `endCol`, inclusive.

For example, if `nightSky` is a `SkyView` object where `view` contains the values shown below, the call `nightSky.getAverage(1, 2, 0, 1)` should return `0.8`. (The average is  $(1.1 + 1.4 + 0.2 + 0.5) / 4$ , which equals `0.8`). The section being averaged is indicated by the dark outline in the table below.

view	0	1	2
0	0.3	0.7	0.8
1	1.1	1.4	0.4
2	0.2	0.5	0.1
3	0.9	0.6	1.6

Class information repeated from the beginning of the question

```
public class SkyView  
  
private double[][] view  
public SkyView(int numRows, int numCols, double[] scanned)  
public double getAverage(int startRow, int endRow,  
                        int startCol, int endCol)
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2013 AP<sup>®</sup> COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `getAverage` below.

```
/** Returns the average of the values in a rectangular section of view.  
 * @param startRow the first row index of the section  
 * @param endRow the last row index of the section  
 * @param startCol the first column index of the section  
 * @param endCol the last column index of the section  
 * Precondition: 0 <= startRow <= endRow < view.length  
 * Precondition: 0 <= startCol <= endCol < view[0].length  
 * @return the average of the values in the specified section of view  
 */  
public double getAverage(int startRow, int endRow,  
                        int startCol, int endCol)
```

**STOP**

**END OF EXAM**



## **AP<sup>®</sup> Computer Science A 2013 Scoring Guidelines**

### **The College Board**

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of over 6,000 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT<sup>®</sup> and the Advanced Placement Program<sup>®</sup>. The organization also serves the education community through research and advocacy on behalf of students, educators, and schools. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2013 The College Board. College Board, Advanced Placement Program, AP, SAT and the acorn logo are registered trademarks of the College Board. All other products and services may be trademarks of their respective owners.

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).  
AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).



# **AP® COMPUTER SCIENCE A**

## **2013 SCORING GUIDELINES**

### **Question 1: SongList**

<b>Part (a)</b>	<code>getDownloadInfo</code>	<b>4 points</b>
-----------------	------------------------------	-----------------

**Intent:** Search download list for requested title and return matching `DownloadInfo` object if found.

- +1 Accesses all necessary entries in `downloadList` (*no bounds errors*)
- +3 Identifies and returns matching entry in `downloadList`, if it exists
  - +1 Calls `getTitle` on `DownloadInfo` object from `downloadList`
- +1 Checks for equality between title from list object and `title` parameter (*must use String equality check*)
- +1 Returns reference to matching object if present; `null` if not (*point not awarded for early return*)

<b>Part (b)</b>	<code>updateDownloads</code>	<b>5 points</b>
-----------------	------------------------------	-----------------

**Intent:** Update `downloadList` with information from list of titles

- +1 Accesses all entries in `titles` (*no bounds error for titles*)
- +1 Calls `getDownloadInfo(title)` to determine whether title from `titles` list exists in `downloadList`
- +1 Increments the count in matching `DownloadInfo` object if title is in `downloadList`
- +1 Constructs new `DownloadInfo` object (with correct information) if title is not in `downloadList` (*point not awarded if incremented at time of construction*)
- +1 Adds constructed object to end of `downloadList` if title is not in `downloadList` (*point not awarded if added more than once*)

<b>Question-Specific Penalties</b>
------------------------------------

- 1 (g) Uses `getLength/getSize` for `ArrayList` size
- 2 (v) Consistently uses incorrect array name instead of `downloadList/titles`
- 1 (z) Attempts to return a value from `updateDownloads`

# **AP® COMPUTER SCIENCE A 2013 SCORING GUIDELINES**

## **Question 2: TokenPass**

<b>Part (a)</b>	TokenPass constructor	<b>4 points</b>
-----------------	-----------------------	-----------------

**Intent:** Create TokenPass object and correctly initialize game state

- +1** Creates instance variable board as int array of size playerCount
- +1** Computes a random number between 1 and 10, inclusive, and a random number between 0 and playerCount-1, inclusive
- +1** Initializes all entries in board with computed random value (*no bounds errors*)
- +1** Initializes instance variable currentPlayer to computed random value

<b>Part (b)</b>	distributeCurrentPlayerTokens	<b>5 points</b>
-----------------	-------------------------------	-----------------

**Intent:** Distribute all tokens from currentPlayer position to subsequent positions in array

- +1** Uses initial value of board[currentPlayer] to control distribution of tokens
- +1** Increases at least one board entry in the context of a loop
- +1** Starts distribution of tokens at correct board entry
- +1** Distributes next token (if any remain) to position 0 after distributing to highest position in board
- +1** On exit: token count at each position in board is correct

<b>Question-Specific Penalties</b>
------------------------------------

- 2** (v) Consistently uses incorrect array name instead of board
- 1** (y) Destruction of persistent data (currentPlayer)
- 1** (z) Attempts to return a value from distributeCurrentPlayerTokens

# **AP® COMPUTER SCIENCE A 2013 SCORING GUIDELINES**

## **Question 4: SkyView**

<b>Part (a)</b>	SkyView constructor	<b>5 points</b>
-----------------	---------------------	-----------------

**Intent:** Construct SkyView object from 1D array of scan data

- +1 Constructs correctly-sized 2D array of doubles and assigns to instance variable view
- +1 Initializes at least one element of view with value from element of scanned (*must be in context of loop*)
- +1 Places consecutive values from scanned into at least one row of view in original order
- +1 Places consecutive values from scanned into at least one row of view in reverse order
- +1 On exit: all elements of view have correct values (*no bounds errors on view or scanned*)

<b>Part (b)</b>	getAverage	<b>4 points</b>
-----------------	------------	-----------------

**Intent:** Compute and return average of rectangular section of view, specified by parameters

- +1 Declares and initializes a double accumulator
- +1 Adds all and only necessary values from view to accumulator (*no bounds errors*)
- +1 Computes average of specified rectangular section
- +1 Returns the computed average (*computation must involve view*)

<b>Question-Specific Penalties</b>
------------------------------------

- 2 (v) Consistently uses incorrect array name instead of view/scanned



## **AP® Computer Science A 2012 Free-Response Questions**

### **About the College Board**

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of more than 5,900 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT® and the Advanced Placement Program®. The organization also serves the education community through research and advocacy on behalf of students, educators, and schools.

© 2012 The College Board. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service and inspiring minds are trademarks owned by the College Board. All other products and services may be trademarks of their respective owners. Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org). Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.org/inquiry/cbpermit.html](http://www.collegeboard.org/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).



**2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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. A mountain climbing club maintains a record of the climbs that its members have made. Information about a climb includes the name of the mountain peak and the amount of time it took to reach the top. The information is contained in the `ClimbInfo` class as declared below.

```
public class ClimbInfo
{
    /** Creates a ClimbInfo object with name peakName and time climbTime.
     *  @param peakName the name of the mountain peak
     *  @param climbTime the number of minutes taken to complete the climb
     */
    public ClimbInfo(String peakName, int climbTime)
    { /* implementation not shown */ }

    /** @return the name of the mountain peak
     */
    public String getName()
    { /* implementation not shown */ }

    /** @return the number of minutes taken to complete the climb
     */
    public int getTime()
    { /* implementation not shown */ }

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

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The ClimbingClub class maintains a list of the climbs made by members of the club. The declaration of the ClimbingClub class is shown below. You will write two different implementations of the addClimb method. You will also answer two questions about an implementation of the distinctPeakNames method.

```
public class ClimbingClub
{
    /**
     * The list of climbs completed by members of the club.
     * Guaranteed not to be null. Contains only non-null references.
     */
    private List<ClimbInfo> climbList;

    /**
     * Creates a new ClimbingClub object.
     */
    public ClimbingClub()
    {   climbList = new ArrayList<ClimbInfo>(); }

    /**
     * Adds a new climb with name peakName and time climbTime to the list of climbs.
     * @param peakName the name of the mountain peak climbed
     * @param climbTime the number of minutes taken to complete the climb
     */
    public void addClimb(String peakName, int climbTime)
    {   /* to be implemented in part (a) with ClimbInfo objects in the order they were added */
        /* to be implemented in part (b) with ClimbInfo objects in alphabetical order by name */
    }

    /**
     * @return the number of distinct names in the list of climbs
     */
    public int distinctPeakNames()
    {   /* implementation shown in part (c) */
    }

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

Part (a) begins on page 4.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write an implementation of the `ClimbingClub` method `addClimb` that stores the `ClimbInfo` objects in the order they were added. This implementation of `addClimb` should create a new `ClimbInfo` object with the given name and time. It appends a reference to that object to the end of `climbList`. For example, consider the following code segment.

```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
```

When the code segment has completed executing, the instance variable `climbList` would contain the following entries.

Peak Name	"Monadnock"	"Whiteface"	"Algonquin"	"Monadnock"
Climb Time	274	301	225	344

Information repeated from the beginning of the question

```
public class ClimbInfo

public ClimbInfo(String peakName, int climbTime)
public String getName()
public int getTime()

public class ClimbingClub

private List<ClimbInfo> climbList
public void addClimb(String peakName, int climbTime)
public int distinctPeakNames()
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `addClimb` below.

```
/** Adds a new climb with name peakName and time climbTime to the list of climbs.  
 * @param peakName the name of the mountain peak climbed  
 * @param climbTime the number of minutes taken to complete the climb  
 * Postcondition: The new entry is at the end of climbList;  
 *                  The order of the remaining entries is unchanged.  
 */  
public void addClimb(String peakName, int climbTime)
```

Part (b) begins on page 6.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write an implementation of the `ClimbingClub` method `addClimb` that stores the elements of `climbList` in alphabetical order by name (as determined by the `compareTo` method of the `String` class). This implementation of `addClimb` should create a new `ClimbInfo` object with the given name and time and then insert the object into the appropriate position in `climbList`. Entries that have the same name will be grouped together and can appear in any order within the group. For example, consider the following code segment.

```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
```

When the code segment has completed execution, the instance variable `climbList` would contain the following entries in either of the orders shown below.

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	344	274	301

OR

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	274	344	301

You may assume that `climbList` is in alphabetical order by name when the method is called. When the method has completed execution, `climbList` should still be in alphabetical order by name.

Information repeated from the beginning of the question

```
public class ClimbInfo

public ClimbInfo(String peakName, int climbTime)
public String getName()
public int getTime()

public class ClimbingClub

private List<ClimbInfo> climbList
public void addClimb(String peakName, int climbTime)
public int distinctPeakNames()
```

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `addClimb` below.

```
/** Adds a new climb with name peakName and time climbTime to the list of climbs.  
 * Alphabetical order is determined by the compareTo method of the String class.  
 * @param peakName the name of the mountain peak climbed  
 * @param climbTime the number of minutes taken to complete the climb  
 * Precondition: entries in climbList are in alphabetical order by name.  
 * Postcondition: entries in climbList are in alphabetical order by name.  
 */  
public void addClimb(String peakName, int climbTime)
```

Part (c) begins on page 8.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (c) The ClimbingClub method `distinctPeakNames` is intended to return the number of different names in `climbList`. For example, after the following code segment has completed execution, the value of the variable `numNames` would be 3.

```
ClimbingClub hikerClub = new ClimbingClub();
hikerClub.addClimb("Monadnock", 274);
hikerClub.addClimb("Whiteface", 301);
hikerClub.addClimb("Algonquin", 225);
hikerClub.addClimb("Monadnock", 344);
int numNames = hikerClub.distinctPeakNames();
```

Consider the following implementation of method `distinctPeakNames`.

```
/** @return the number of distinct names in the list of climbs */
public int distinctPeakNames()
{
    if (climbList.size() == 0)
    {
        return 0;
    }

    ClimbInfo currInfo = climbList.get(0);
    String prevName = currInfo.getName();
    String currName = null;
    int numNames = 1;

    for (int k = 1; k < climbList.size(); k++)
    {
        currInfo = climbList.get(k);
        currName = currInfo.getName();
        if (prevName.compareTo(currName) != 0)
        {
            numNames++;
            prevName = currName;
        }
    }
    return numNames;
}
```

## **2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Assume that `addClimb` works as specified, regardless of what you wrote in parts (a) and (b).

- (i) Does this implementation of the `distinctPeakNames` method work as intended when the `addClimb` method stores the `ClimbInfo` objects in the order they were added as described in part (a)?

Circle one of the answers below.

YES

NO

- (ii) Does this implementation of the `distinctPeakNames` method work as intended when the `addClimb` method stores the `ClimbInfo` objects in alphabetical order by name as described in part (b)?

Circle one of the answers below.

YES

NO

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. Consider a software system that models a horse barn. Classes that represent horses implement the following interface.

```
public interface Horse
{
    /** @return the horse's name */
    String getName();

    /** @return the horse's weight */
    int getWeight();

    // There may be methods that are not shown.
}
```

A horse barn consists of  $N$  numbered spaces. Each space can hold at most one horse. The spaces are indexed starting from 0; the index of the last space is  $N - 1$ . No two horses in the barn have the same name.

The declaration of the `HorseBarn` class is shown below. You will write two unrelated methods of the `HorseBarn` class.

```
public class HorseBarn
{
    /** The spaces in the barn. Each array element holds a reference to the horse
     * that is currently occupying the space. A null value indicates an empty space.
     */
    private Horse[] spaces;

    /**
     * Returns the index of the space that contains the horse with the specified name.
     * Precondition: No two horses in the barn have the same name.
     * @param name the name of the horse to find
     * @return the index of the space containing the horse with the specified name;
     *         -1 if no horse with the specified name is in the barn.
     */
    public int findHorseSpace(String name)
    { /* to be implemented in part (a) */ }

    /**
     * Consolidates the barn by moving horses so that the horses are in adjacent spaces,
     * starting at index 0, with no empty space between any two horses.
     * Postcondition: The order of the horses is the same as before the consolidation.
     */
    public void consolidate()
    { /* to be implemented in part (b) */ }

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

Part (a) begins on page 14.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the HorseBarn method `findHorseSpace`. This method returns the index of the space in which the horse with the specified name is located. If there is no horse with the specified name in the barn, the method returns -1.

For example, assume a `HorseBarn` object called `sweetHome` has horses in the following spaces.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	"Lady" 1575	null	"Patches" 1350	"Duke" 1410

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

Method Call	Value Returned	Reason
<code>sweetHome.findHorseSpace("Trigger")</code>	0	A horse named Trigger is in space 0.
<code>sweetHome.findHorseSpace("Silver")</code>	2	A horse named Silver is in space 2.
<code>sweetHome.findHorseSpace("Coco")</code>	-1	A horse named Coco is not in the barn.

Information repeated from the beginning of the question

```
public interface Horse  
  
String getName()  
int getWeight()  
  
public class HorseBarn  
  
private Horse[] spaces  
public int findHorseSpace(String name)  
public void consolidate()
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `findHorseSpace` below.

```
/** Returns the index of the space that contains the horse with the specified name.  
 * Precondition: No two horses in the barn have the same name.  
 * @param name the name of the horse to find  
 * @return the index of the space containing the horse with the specified name;  
 *         -1 if no horse with the specified name is in the barn.  
 */  
public int findHorseSpace(String name)
```

Part (b) begins on page 16.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the HorseBarn method `consolidate`. This method consolidates the barn by moving horses so that the horses are in adjacent spaces, starting at index 0, with no empty spaces between any two horses.

After the barn is consolidated, the horses are in the same order as they were before the consolidation.

For example, assume a barn has horses in the following spaces.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	null	null	"Patches" 1350	"Duke" 1410

The following table shows the arrangement of the horses after `consolidate` is called.

0	1	2	3	4	5	6
"Trigger" 1340	"Silver" 1210	"Patches" 1350	"Duke" 1410	null	null	null

Information repeated from the beginning of the question

```
public interface Horse  
  
String getName()  
int getWeight()  
  
public class HorseBarn  
  
private Horse[] spaces  
public int findHorseSpace(String name)  
public void consolidate()
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method consolidate below.

```
/** Consolidates the barn by moving horses so that the horses are in adjacent spaces,  
 * starting at index 0, with no empty space between any two horses.  
 * Postcondition: The order of the horses is the same as before the consolidation.  
 */  
public void consolidate()
```

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. A grayscale image is represented by a 2-dimensional rectangular array of pixels (picture elements). A pixel is an integer value that represents a shade of gray. In this question, pixel values can be in the range from 0 through 255, inclusive. A black pixel is represented by 0, and a white pixel is represented by 255.

The declaration of the `GrayImage` class is shown below. You will write two unrelated methods of the `GrayImage` class.

```
public class GrayImage
{
    public static final int BLACK = 0;
    public static final int WHITE = 255;

    /**
     * The 2-dimensional representation of this image. Guaranteed not to be null.
     * All values in the array are within the range [BLACK, WHITE], inclusive.
     */
    private int[][] pixelValues;

    /**
     * @return the total number of white pixels in this image.
     * Postcondition: this image has not been changed.
     */
    public int countWhitePixels()
    { /* to be implemented in part (a) */ }

    /**
     * Processes this image in row-major order and decreases the value of each pixel at
     * position (row, col) by the value of the pixel at position (row + 2, col + 2) if it exists.
     * Resulting values that would be less than BLACK are replaced by BLACK.
     * Pixels for which there is no pixel at position (row + 2, col + 2) are unchanged.
     */
    public void processImage()
    { /* to be implemented in part (b) */ }
}
```

- (a) Write the method `countWhitePixels` that returns the number of pixels in the image that contain the value `WHITE`. For example, assume that `pixelValues` contains the following image.

	0	1	2	3	4
0	255	184	178	84	129
1	84	255	255	130	84
2	78	255	0	0	78
3	84	130	255	130	84

A call to `countWhitePixels` method would return 5 because there are 5 entries (shown in boldface) that have the value `WHITE`.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `countWhitePixels` below.

```
/** @return the total number of white pixels in this image.  
 *  Postcondition: this image has not been changed.  
 */  
public int countWhitePixels()
```

Part (b) begins on page 20.

## 2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the method `processImage` that modifies the image by changing the values in the instance variable `pixelValues` according to the following description. The pixels in the image are processed one at a time in row-major order. Row-major order processes the first row in the array from left to right and then processes the second row from left to right, continuing until all rows are processed from left to right. The first index of `pixelValues` represents the row number, and the second index represents the column number.

The pixel value at position (row, col) is decreased by the value at position (row + 2, col + 2) if such a position exists. If the result of the subtraction is less than the value `BLACK`, the pixel is assigned the value of `BLACK`. The values of the pixels for which there is no pixel at position (row + 2, col + 2) remain unchanged. You may assume that all the original values in the array are within the range `[BLACK, WHITE]`, inclusive.

The following diagram shows the contents of the instance variable `pixelValues` before and after a call to `processImage`. The values shown in boldface represent the pixels that could be modified in a grayscale image with 4 rows and 5 columns.

Before Call to <code>processImage</code>						After Call to <code>processImage</code>					
	0	1	2	3	4		0	1	2	3	4
0	<b>221</b>	<b>184</b>	<b>178</b>	84	135	0	<b>221</b>	<b>184</b>	<b>100</b>	84	135
1	<b>84</b>	<b>255</b>	<b>255</b>	130	84	1	<b>0</b>	<b>125</b>	<b>171</b>	130	84
2	78	255	0	0	78	2	78	255	0	0	78
3	84	130	255	130	84	3	84	130	255	130	84

Information repeated from the beginning of the question

```
public class GrayImage  
  
public static final int BLACK = 0  
public static final int WHITE = 255  
private int[][] pixelValues  
public int countWhitePixels()  
public void processImage()
```

**WRITE YOUR SOLUTION ON THE NEXT PAGE.**

## **2012 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `processImage` below.

```
/** Processes this image in row-major order and decreases the value of each pixel at
 * position (row, col) by the value of the pixel at position (row + 2, col + 2) if it exists.
 * Resulting values that would be less than BLACK are replaced by BLACK.
 * Pixels for which there is no pixel at position (row + 2, col + 2) are unchanged.
 */
public void processImage()
```

**STOP**

**END OF EXAM**



## **AP® Computer Science A 2012 Scoring Guidelines**

### **The College Board**

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of more than 5,900 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT® and the Advanced Placement Program®. The organization also serves the education community through research and advocacy on behalf of students, educators, and schools. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2012 The College Board. College Board, Advanced Placement Program, AP, SAT and the acorn logo are registered trademarks of the College Board. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.org](http://apcentral.collegeboard.org).



# AP® COMPUTER SCIENCE A 2012 GENERAL SCORING GUIDELINES

Apply the question-specific rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question-specific rubric. No part of a question — (a), (b), or (c) — may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in different parts of that question.

## 1-Point Penalty

- (w) Extraneous code that causes a side effect or prevents earning points in the rubric  
*(e.g., information written to output)*
- (x) Local variables used but none declared
- (y) Destruction of persistent data (*e.g., changing value referenced by parameter*)
- (z) `Void` method or constructor that returns a value

## No Penalty

- o Extraneous code that causes no side effect
- o Extraneous code that is unreachable and would not have earned points in rubric
- o Spelling/case discrepancies where there is no ambiguity\*
- o Local variable not declared, provided that other variables are declared in some part
- o `private` qualifier on local variable
- o Missing `public` qualifier on class or constructor header
- o Keyword used as an identifier
- o Common mathematical symbols used for operators ( $x \bullet \div \leq \geq < > \neq$ )
- o `[]` vs. `()` vs. `<>`
- o `=` instead of `==` (and vice versa)
- o Array/collection element access confusion (`[]` vs. `get` for r-values)
- o Array/collection element modification confusion (`[]` vs. `set` for l-values)
- o `length/size` confusion for array, `String`, and `ArrayList`, with or without `()`
- o Extraneous `[]` when referencing entire array
- o `[i, j]` instead of `[i] [j]`
- o Extraneous size in array declaration, (*e.g.*, `int[size]` `nums = new int[size];`)
- o Missing `;` provided that line breaks and indentation clearly convey intent
- o Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- o Missing `( )` on parameter-less method or constructor invocations
- o Missing `( )` around `if/while` conditions
- o Use of local variable outside declared scope (must be within same method body)
- o Failure to cast object retrieved from nongeneric collection

\* Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be **unambiguously** inferred from context; for example, “`ArrayList`” instead of “`ArrayList`”. As a counterexample, note that if the code declares “`Bug bug;`” and then uses “`Bug.move()`” instead of “`bug.move()`”, the context does **not** allow for the reader to assume the object instead of the class.

# **AP® COMPUTER SCIENCE A 2012 SCORING GUIDELINES**

## **Question 1: Climbing Club**

<b>Part (a)</b>	<code>addClimb (append)</code>	<b>2 points</b>
-----------------	--------------------------------	-----------------

**Intent:** Create new `ClimbInfo` using data from parameters and append to `climbList`

- +1 Creates new `ClimbInfo` object using parametric data correctly
- +1 Appends the created object to `climbList`  
*(no bounds error and no destruction of existing data)  
(point not awarded if inserted more than once)*

<b>Part (b)</b>	<code>addClimb (alphabetical)</code>	<b>6 points</b>
-----------------	--------------------------------------	-----------------

**Intent:** Create new `ClimbInfo` object using data from parameters and insert into `climbList`, maintaining alphabetical order

- +1 Creates new `ClimbInfo` object(s), using parametric data correctly
- +1 Compares `peakName` value with value retrieved from object in list (*must use getName*)
- +1 Inserts object into list based on a comparison (other than equality) with object in list  
*(point not awarded if inserted more than once)*
- +1 Compares parametric data with all appropriate entries in `climbList` (*no bounds error*)
- +1 Inserts new `ClimbInfo` object into `climbList` (*no destruction of existing data*)
- +1 Inserts new `ClimbInfo` object into `climbList` once and only once in maintaining alphabetical order (*no destruction of existing data*)

<b>Part (c)</b>	<code>analysis</code>	<b>1 point</b>
-----------------	-----------------------	----------------

**Intent:** Analyze behavioral differences between **append** and **alphabetical** versions of `addClimb`

- +1 (i) NO (ii) YES Both must be answered correctly

<b>Question-Specific Penalties</b>
------------------------------------

- 1 (z) Attempts to return a value from `addClimb`

# **AP® COMPUTER SCIENCE A 2012 SCORING GUIDELINES**

## **Question 3: Horse Barn**

<b>Part (a)</b>	<code>findHorseSpace</code>	<b>4 points</b>
-----------------	-----------------------------	-----------------

**Intent:** Return index of space containing horse with specified name

- +1 Accesses all entries in spaces (*no bounds errors*)
- +1 Checks for null reference in array and avoids dereferencing it (*in context of loop*)
- +1 Checks for name equality between array element and parameter (*must use String equality check*)
- +1 Returns correct index, if present; -1 point if not

<b>Part (b)</b>	<code>consolidate</code>	<b>5 points</b>
-----------------	--------------------------	-----------------

**Intent:** Repopulate spaces such that the order of all non-null entries is preserved and all null entries are found contiguously at the largest indices

- +1 Accesses all entries in spaces (*no bounds errors*)
- +1 Identifies and provides different treatment of null and non-null elements in array
- +1 Assigns element in array to a smaller index (*must have identified source as non-null or destination as null*)
- +1 On exit: The number, integrity, and order of all identified non-null elements in spaces is preserved, and the number of null elements is preserved
- +1 On exit: All non-null elements in spaces are in contiguous locations, beginning at index 0 (*no destruction of data*)

### **Question-Specific Penalties**

- 1 (z) Attempts to return a value from consolidate
- 2 (v) Consistently uses incorrect array name instead of spaces

# **AP® COMPUTER SCIENCE A 2012 SCORING GUIDELINES**

## **Question 4: GrayImage**

<b>Part (a)</b>	<code>countWhitePixels</code>	<b>4 points</b>
-----------------	-------------------------------	-----------------

**Intent:** Return the number of white pixels in the image

- +1 Accesses all entries in `pixelValues` (*no bounds errors*)
- +1 Compares an entry of array with `WHITE` or with 255 in context of iteration
- +1 Initializes and increments a counter
- +1 Returns correct count of number of white pixels in `pixelValues`

<b>Part (b)</b>	<code>processImage</code>	<b>5 points</b>
-----------------	---------------------------	-----------------

**Intent:** Process elements of `pixelValues` and apply specified formula

- +1 Accesses all necessary elements in at least one row or one column of `pixelValues`
- +1 Accesses all necessary elements of `pixelValues` (*no bounds errors*)
- +1 Decrements element at index `[a][b]` by the original value found in element at index `[a+2][b+2]`
- +1 Modifies all and only appropriate elements of `pixelValues`  
*(changes must not affect last two rows and columns)*
- +1 Assigns `BLACK` or 0 to elements of `pixelValues` that would otherwise have a value less than `BLACK` (negative value)

<b>Question-Specific Penalties</b>
------------------------------------

- 1 (z) Attempts to return a value from `processImage`



## **AP® Computer Science A 2011 Free-Response Questions**

### **About the College Board**

The College Board is a mission-driven not-for-profit organization that connects students to college success and opportunity. Founded in 1900, the College Board was created to expand access to higher education. Today, the membership association is made up of more than 5,900 of the world's leading educational institutions and is dedicated to promoting excellence and equity in education. Each year, the College Board helps more than seven million students prepare for a successful transition to college through programs and services in college readiness and college success — including the SAT® and the Advanced Placement Program®. The organization also serves the education community through research and advocacy on behalf of students, educators and schools.

© 2011 The College Board. College Board, Advanced Placement Program, AP, AP Central, SAT and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service and inspiring minds are trademarks owned by the College Board. All other products and services may be trademarks of their respective owners. Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org). Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.org/inquiry/cbpermit.html](http://www.collegeboard.org/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 Quick Reference found in the Appendix 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. Digital sounds can be represented as an array of integer values. For this question, you will write two unrelated methods of the `Sound` class.

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

```
public class Sound
{
    /** the array of values in this sound; guaranteed not to be null */
    private int[] samples;

    /** Changes those values in this sound that have an amplitude greater than limit.
     *  Values greater than limit are changed to limit.
     *  Values less than -limit are changed to -limit.
     *  @param limit the amplitude limit
     *  Precondition: limit ≥ 0
     *  @return the number of values in this sound that this method changed
    */
    public int limitAmplitude(int limit)
    { /* to be implemented in part (a) */ }

    /** Removes all silence from the beginning of this sound.
     *  Silence is represented by a value of 0.
     *  Precondition: samples contains at least one nonzero value
     *  Postcondition: the length of samples reflects the removal of starting silence
    */
    public void trimSilenceFromBeginning()
    { /* to be implemented in part (b) */ }

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

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) The volume of a sound depends on the amplitude of each value in the sound. The amplitude of a value is its absolute value. For example, the amplitude of -2300 is 2300 and the amplitude of 4000 is 4000.

Write the method `limitAmplitude` that will change any value that has an amplitude greater than the given limit. Values that are greater than `limit` are replaced with `limit`, and values that are less than `-limit` are replaced with `-limit`. The method returns the total number of values that were changed in the array. For example, assume that the array `samples` has been initialized with the following values.

40	2532	17	-2300	-17	-4000	2000	1048	-420	33	15	-32	2030	3223
----	------	----	-------	-----	-------	------	------	------	----	----	-----	------	------

When the statement

```
int numChanges = limitAmplitude(2000);
```

is executed, the value of `numChanges` will be 5, and the array `samples` will contain the following values.

40	2000	17	-2000	-17	-2000	2000	1048	-420	33	15	-32	2000	2000
----	------	----	-------	-----	-------	------	------	------	----	----	-----	------	------

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.  
 * Values greater than limit are changed to limit.  
 * Values less than -limit are changed to -limit.  
 * @param limit the amplitude limit  
 *      Precondition: limit ≥ 0  
 *      @return the number of values in this sound that this method changed  
 */  
public int limitAmplitude(int limit)
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Recorded sound often begins with silence. Silence in a sound is represented by a value of 0.

Write the method `trimSilenceFromBeginning` that removes the silence from the beginning of a sound. To remove starting silence, a new array of values is created that contains the same values as the original `samples` array in the same order but without the leading zeros. The instance variable `samples` is updated to refer to the new array. For example, suppose the instance variable `samples` refers to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	0	0	0	-14	0	-35	-39	0	-7	16	32	37	29	0	0

After `trimSilenceFromBeginning` has been called, the instance variable `samples` will refer to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	-14	0	-35	-39	0	-7	16	32	37	29	0	0

Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.  
 * Silence is represented by a value of 0.  
 * Precondition: samples contains at least one nonzero value  
 * Postcondition: the length of samples reflects the removal of starting silence  
 */  
public void trimSilenceFromBeginning()
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. A fuel depot has a number of fuel tanks arranged in a line and a robot that moves a filling mechanism back and forth along the line so that the tanks can be filled. A fuel tank is specified by the `FuelTank` interface below.

```
public interface FuelTank
{
    /** @return an integer value that ranges from 0 (empty) to 100 (full) */
    int getFuelLevel();
}
```

A fuel depot keeps track of the fuel tanks and the robot. The following figure represents the tanks and the robot in a fuel depot. The robot, indicated by the arrow, is currently at index 2 and is facing to the right.

Tank index	0	1	2	3	4	5
Fuel level in tank	80	70	20	45	50	25
Robot	→					
<hr/> <hr/>						

The state of the robot includes the index of its location and the direction in which it is facing (to the right or to the left). This information is specified in the `FuelRobot` interface as shown in the following declaration.

```
public interface FuelRobot
{
    /** @return the index of the current location of the robot */
    int getCurrentIndex();

    /** Determine whether the robot is currently facing to the right
     *  @return true if the robot is facing to the right (toward tanks with larger indexes)
     *          false if the robot is facing to the left (toward tanks with smaller indexes)
     */
    boolean isFacingRight();

    /** Changes the current direction of the robot */
    void changeDirection();

    /** Moves the robot in its current direction by the number of locations specified.
     *  @param numLocs the number of locations to move. A value of 1 moves
     *                  the robot to the next location in the current direction.
     *  @Precondition: numLocs > 0
     */
    void moveForward(int numLocs);
}
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

A fuel depot is represented by the `FuelDepot` class as shown in the following class declaration.

```
public class FuelDepot
{
    /** The robot used to move the filling mechanism */
    private FuelRobot filler;

    /** The list of fuel tanks */
    private List<FuelTank> tanks;

    /** Determines and returns the index of the next tank to be filled.
     * @param threshold fuel tanks with a fuel level ≤ threshold may be filled
     * @return index of the location of the next tank to be filled
     * Postcondition: the state of the robot has not changed
     */
    public int nextTankToFill(int threshold)
    { /* to be implemented in part (a) */ }

    /** Moves the robot to location locIndex.
     * @param locIndex the index of the location of the tank to move to
     * Precondition: 0 ≤ locIndex < tanks.size()
     * Postcondition: the current location of the robot is locIndex
     */
    public void moveToLocation(int locIndex)
    { /* to be implemented in part (b) */ }

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

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

(a) Write the `FuelDepot` method `nextTankToFill` that returns the index of the next tank to be filled.

The index for the next tank to be filled is determined according to the following rules:

- Return the index of a tank with the lowest fuel level that is less than or equal to a given threshold.  
If there is more than one fuel tank with the same lowest fuel level, any of their indexes can be returned.
- If there are no tanks with a fuel level less than or equal to the threshold, return the robot's current index.

For example, suppose the tanks contain the fuel levels shown in the following figure.

Tank index	0	1	2	3	4	5	6
Fuel level in tank	20	30	80	55	50	75	20
Robot	→						

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

threshold	Return Value	Rationale
50	0 or 6	20 is the lowest fuel level, so either 0 or 6 can be returned.
15	2	There are no tanks with a fuel level $\leq$ threshold, so the robot's current index is returned.

Complete method `nextTankToFill` below.

```
/** Determines and returns the index of the next tank to be filled.
 * @param threshold fuel tanks with a fuel level ≤ threshold may be filled
 * @return index of the location of the next tank to be filled
 * Postcondition: the state of the robot has not changed
 */
public int nextTankToFill(int threshold)
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the FuelDepot method `moveToLocation` that will move the robot to the given tank location. Because the robot can only move forward, it may be necessary to change the direction of the robot before having it move. Do not move the robot past the end of the line of fuel tanks.

Complete method `moveToLocation` below.

```
/** Moves the robot to location locIndex.  
 *  @param locIndex the index of the location of the tank to move to  
 *  Precondition: 0 ≤ locIndex < tanks.size()  
 *  Postcondition: the current location of the robot is locIndex  
 */  
public void moveToLocation(int locIndex)
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. In this question you will write two methods for a class `RouteCipher` that encrypts (puts into a coded form) a message by changing the order of the characters in the message. The route cipher fills a two-dimensional array with single-character substrings of the original message in row-major order, encrypting the message by retrieving the single-character substrings in column-major order.

For example, the word "Surprise" can be encrypted using a 2-row, 4-column array as follows.

Original Message	Contents of Array	Encrypted Message								
"Surprise"	$\rightarrow$ <table border="1" style="margin: auto;"><tr><td>"S"</td><td>"u"</td><td>"r"</td><td>"p"</td></tr><tr><td>"r"</td><td>"i"</td><td>"s"</td><td>"e"</td></tr></table> $\rightarrow$	"S"	"u"	"r"	"p"	"r"	"i"	"s"	"e"	"Sruirspe"
"S"	"u"	"r"	"p"							
"r"	"i"	"s"	"e"							

An incomplete implementation of the `RouteCipher` class is shown below.

```
public class RouteCipher
{
    /** A two-dimensional array of single-character strings, instantiated in the constructor */
    private String[][] letterBlock;

    /** The number of rows of letterBlock, set by the constructor */
    private int numRows;

    /** The number of columns of letterBlock, set by the constructor */
    private int numCols;

    /** Places a string into letterBlock in row-major order.
     * @param str the string to be processed
     * Postcondition:
     *   if str.length() < numRows * numCols, "A" is placed in each unfilled cell
     *   if str.length() > numRows * numCols, trailing characters are ignored
     */
    private void fillBlock(String str)
    { /* to be implemented in part (a) */ }

    /** Extracts encrypted string from letterBlock in column-major order.
     * Precondition: letterBlock has been filled
     * @return the encrypted string from letterBlock
     */
    private String encryptBlock()
    { /* implementation not shown */ }

    /** Encrypts a message.
     * @param message the string to be encrypted
     * @return the encrypted message;
     *         if message is the empty string, returns the empty string
     */
    public String encryptMessage(String message)
    { /* to be implemented in part (b) */ }

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

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the method `fillBlock` that fills the two-dimensional array `letterBlock` with one-character strings from the string passed as parameter `str`.

The array must be filled in row-major order—the first row is filled from left to right, then the second row is filled from left to right, and so on, until all rows are filled.

If the length of the parameter `str` is smaller than the number of elements of the array, the string "A" is placed in each of the unfilled cells. If the length of `str` is larger than the number of elements in the array, the trailing characters are ignored.

For example, if `letterBlock` has 3 rows and 5 columns and `str` is the string "Meet at noon", the resulting contents of `letterBlock` would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"n"	"o"
"o"	"n"	"A"	"A"	"A"

If `letterBlock` has 3 rows and 5 columns and `str` is the string "Meet at midnight", the resulting contents of `letterBlock` would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"m"	"i"
"d"	"n"	"i"	"g"	"h"

The following expression may be used to obtain a single-character string at position `k` of the string `str`.

`str.substring(k, k + 1)`

Complete method `fillBlock` below.

```
/** Places a string into letterBlock in row-major order.  
 * @param str the string to be processed  
 * Postcondition:  
 *   if str.length() < numRows * numCols, "A" is placed in each unfilled cell  
 *   if str.length() > numRows * numCols, trailing characters are ignored  
 */  
private void fillBlock(String str)
```

## 2011 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the method `encryptMessage` that encrypts its string parameter `message`. The method builds an encrypted version of `message` by repeatedly calling `fillBlock` with consecutive, nonoverlapping substrings of `message` and concatenating the results returned by a call to `encryptBlock` after each call to `fillBlock`. When all of `message` has been processed, the concatenated string is returned. Note that if `message` is the empty string, `encryptMessage` returns an empty string.

The following example shows the process carried out if `letterBlock` has 2 rows and 3 columns and `encryptMessage("Meet at midnight")` is executed.

Substring	letterBlock after Call to <code>fillBlock</code>	Value Returned by <code>encryptBlock</code>	Concatenated String						
"Meet a"	<table border="1"><tr><td>"M"</td><td>"e"</td><td>"e"</td></tr><tr><td>"t"</td><td>" "</td><td>"a"</td></tr></table>	"M"	"e"	"e"	"t"	" "	"a"	"Mte ea"	"Mte ea"
"M"	"e"	"e"							
"t"	" "	"a"							
"t midn"	<table border="1"><tr><td>"t"</td><td>" "</td><td>"m"</td></tr><tr><td>"i"</td><td>"d"</td><td>"n"</td></tr></table>	"t"	" "	"m"	"i"	"d"	"n"	"ti dmn"	"Mte eati dmn"
"t"	" "	"m"							
"i"	"d"	"n"							
"ight"	<table border="1"><tr><td>"i"</td><td>"g"</td><td>"h"</td></tr><tr><td>"t"</td><td>"A"</td><td>"A"</td></tr></table>	"i"	"g"	"h"	"t"	"A"	"A"	"itgAhA"	"Mte eati dmnitgAhA"
"i"	"g"	"h"							
"t"	"A"	"A"							

In this example, the method returns the string "Mte eati dmnitgAhA".

Assume that `fillBlock` and `encryptBlock` methods work as specified. Solutions that reimplement the functionality of one or both of these methods will not receive full credit.

Complete method `encryptMessage` below.

```
/** Encrypts a message.  
 * @param message the string to be encrypted  
 * @return the encrypted message;  
 *         if message is the empty string, returns the empty string  
 */  
public String encryptMessage(String message)
```

**STOP**

**END OF EXAM**



## **AP® Computer Science A 2011 Scoring Guidelines**

### **The College Board**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the College Board is composed of more than 5,700 schools, colleges, universities and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,800 colleges through major programs and services in college readiness, college admission, guidance, assessment, financial aid and enrollment. Among its widely recognized programs are the SAT®, the PSAT/NMSQT®, the Advanced Placement Program® (AP®), SpringBoard® and ACCUPLACER®. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities and concerns.

© 2011 The College Board. College Board, ACCUPLACER, Advanced Placement Program, AP, AP Central, SAT, SpringBoard and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service is a trademark owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.org](http://www.collegeboard.org).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

# AP® COMPUTER SCIENCE A 2011 GENERAL SCORING GUIDELINES

**Apply the question-specific rubric first; the question-specific rubric always takes precedence.**

**Penalties:** The penalty categorization below is for cases not covered by the question-specific rubric. Points can only be deducted in a part of the question that has earned credit via the question-specific rubric, and no section may have a negative point total. A given penalty can be assessed **only once** in a question, even if it occurs on different parts of that question. A maximum of 3 penalty points may be assessed over the entire question.

## Nonpenalized Errors

spelling/case discrepancies if no ambiguity\*

local variable not declared if other variables are declared in some part

use of keyword as identifier

[ ] vs. ( ) vs. <>

= instead of == (and vice versa)

length/size confusion for array, String, and ArrayList, with or without ()

private qualifier on local variable

extraneous code with no side effect; e.g., precondition check

common mathematical symbols for operators ( $x \bullet \div \leq \geq < > \neq$ )

missing {} where indentation clearly conveys intent and {} used elsewhere

default constructor called without parens;  
e.g., new Critter;

missing ( ) on parameter-less method call

missing ( ) around if/while conditions

missing ; when majority are present

missing public on class or constructor header

extraneous [ ] when referencing entire array

[i, j] instead of [i][j]

extraneous size in array declaration,  
e.g., int[size] nums = new int[size];

## Minor Errors (½ point)

confused identifier (e.g., len for length or left() for getLeft())

local variables used but none declared

missing new in constructor call

modifying a constant (final)

use of equals or compareTo method on primitives, e.g., int x; ...x.equals(val)

array/collection access confusion ([ ] get)

assignment dyslexia,  
e.g., x + 3 = y; for y = x + 3;

super(method()) instead of super.method()

formal parameter syntax (with type) in method call, e.g., a = method(int x)

missing public from method header when required

"false"/"true" or 0/1 for boolean values

"null" for null

## Major Errors (1 point)

extraneous code that causes side effect; e.g., information written to output

interface or class name instead of variable identifier; e.g., Bug.move() instead of aBug.move()

aMethod(obj) instead of obj.aMethod()

attempt to use private data or method when not accessible

destruction of persistent data (e.g., changing value referenced by parameter)

use of class name in place of super in constructor or method call

void method (or constructor) returns a value

### Applying Minor Penalties

(½ point):

A minor infraction that occurs **exactly once** when the same concept is **correct two or more times** is regarded as an oversight and **not penalized**. A minor penalty **must be assessed** if the item is the **only instance, one of two**, or occurs **two or more times**.

\* Spelling and case discrepancies for identifiers fall under the “nonpenalized” category only if the correction can be **unambiguously** inferred from context; for example, “ArrayList” instead of “ArrayList”. As a counterexample, note that if a student declares “Bug bug;” then uses “Bug.move( )” instead of “bug.move( )”, the context does **not** allow for the reader to assume the object instead of the class.

# AP® COMPUTER SCIENCE A 2011 SCORING GUIDELINES

## Question 1: Sound

<b>Part (a)</b>	limitAmplitude	<b>4½ points</b>
-----------------	----------------	------------------

*Intent:* Change elements of samples that exceed ±limit; return number of changes made

- +3 Identify elements of samples to be modified and modify as required
  - +1 Consider elements of samples
    - +½ Accesses more than one element of samples
    - +½ Accesses every element of samples (*no bounds errors*)
  - +2 Identify and change elements of samples
    - +½ Compares an element of samples with limit
    - +½ Changes at least one element to limit or -limit
    - +1 Changes all and only elements that exceed ±limit to limit or -limit appropriately
- +1½ Calculate and return number of changed elements of samples
  - +1 Initializes and updates a counter to achieve correct number of changed samples
  - +½ Returns value of an updated counter (*requires array access*)

<b>Part (b)</b>	trimSilenceFromBeginning	<b>4½ points</b>
-----------------	--------------------------	------------------

*Intent:* Remove leading elements of samples that have value of 0, potentially resulting in array of different length

- +1½ Identify leading-zero-valued elements of samples
  - +½ Accesses every leading-zero element of samples
  - +½ Compares 0 and an element of samples
  - +½ Compares 0 and multiple elements of samples
- +1 Create array of proper length
  - +½ Determines correct number of elements to be in resulting array
  - +½ Creates new array of determined length
- +2 Remove silence values from samples
  - +½ Copies some values other than leading-zero values
  - +1 Copies all and only values other than leading-zero values, preserving original order
  - +½ Modifies instance variable samples to reference newly created array

<b>Question-Specific Penalties</b>
------------------------------------

- 1 Array identifier confusion (e.g., value instead of samples)
- ½ Array/collection modifier confusion (e.g., using set)

# AP® COMPUTER SCIENCE A 2011 SCORING GUIDELINES

## Question 3: Fuel Depot

Part (a)	nextTankToFill	5 points
<i>Intent:</i> Return index of tank with minimum level ( $\leq$ threshold)		
<b>+4</b> Determine minimum element of tanks that is $\leq$ threshold, if any		
<b>+1½</b> Consider fuel levels of elements of tanks		
<b>+½</b> Accesses fuel level of an element of tanks		
<b>+½</b> Accesses at least one element of tanks in context of repetition (iteration/recursion)		
<b>+½</b> Accesses every element of tanks at least once		
<b>+2½</b> Identify minimum element of tanks that is $\leq$ threshold		
<b>+½</b> Compares fuel levels from at least two elements of tanks		
<b>+½</b> Implements algorithm to find minimum		
<b>+½</b> Identifies tank (object or index) holding identified minimum		
<b>+½</b> Compares threshold with fuel level from at least one element of tanks		
<b>+½</b> Determines element identified as minimum fuel level that is also $\leq$ threshold		
<b>+1</b> Return the index of the element satisfying the conditions, or the current index if no element does so		
<b>+½</b> Returns index of element identified as satisfying threshold & minimum conditions*		
<b>+½</b> Returns <code>filler.getCurrentIndex()</code> when no element satisfies conditions*		

\*Note: Point is not awarded if wrong data type is returned.

Part (b)	moveToLocation	4 points
<i>Intent:</i> Move robot to given tank location		
<b>+2</b> Ensure robot is pointing in direction of tank to be filled		
<b>+½</b> Determines direction <code>filler</code> is currently facing		
<b>+½</b> Changes <code>filler</code> 's direction for some condition		
<b>+1</b> Establishes <code>filler</code> 's direction as appropriate for all conditions		
<b>+2</b> Place robot at specified location		
<b>+½</b> Invokes <code>moveForward</code> method with a parameter		
<b>+½</b> Invokes <code>moveForward</code> method with a verified non-zero parameter		
<b>+1</b> Invokes <code>filler.moveForward</code> method with a correctly computed parameter		

# AP® COMPUTER SCIENCE A 2011 SCORING GUIDELINES

## Question 4: Cipher

<b>Part (a)</b>	<code>fillBlock</code>	<b>3½ points</b>
-----------------	------------------------	------------------

*Intent:* Fill `letterBlock` in row-major order from parameter; pad block or truncate string as needed

- +½ Copies at least one substring from parameter to `letterBlock`
- +½ Completely fills `letterBlock` from parameter if possible  
(no bounds errors in `letterBlock` or parameter)
- +1 Results in a distribution of all consecutive one-character substrings from parameter to `letterBlock` (ignores surplus characters)
- +½ Copies these one-character substrings from parameter to `letterBlock` in such a way that the result is in row-major order
- +1 Pads `letterBlock` with "A" if and only if parameter is shorter than block size

<b>Part (b)</b>	<code>encryptMessage</code>	<b>5½ points</b>
-----------------	-----------------------------	------------------

*Intent:* Return encrypted string created by repeatedly invoking `fillBlock` and `encryptBlock` on substrings of parameter and concatenating the results

- +2 Partition parameter
  - +½ Returns the empty string if the parameter is the empty string
  - +½ Creates substrings of parameter that progress through the parameter string (can overlap or skip)
  - +1 Processes every character in parameter exactly once (no bounds errors)
- +3 Fill and encrypt a block, concatenate results
  - +½ Invokes `fillBlock` with parameter or substring of parameter
  - +½ Invokes `fillBlock` on more than one substring of parameter
  - +½ Invokes `encryptBlock` after each invocation of `fillBlock`
  - +½ Concatenates encrypted substrings of parameter
  - +1 Builds complete, encrypted message
- +½ Return resulting built string

<b>Question-Specific Penalties</b>
------------------------------------

- ½ Use of identifier with no apparent resemblance to `letterBlock` for two-dimensional array



## **AP® Computer Science A 2010 Free-Response Questions**

### **The College Board**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the College Board is composed of more than 5,700 schools, colleges, universities and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,800 colleges through major programs and services in college readiness, college admission, guidance, assessment, financial aid and enrollment. Among its widely recognized programs are the SAT®, the PSAT/NMSQT®, the Advanced Placement Program® (AP®), SpringBoard® and ACCUPLACER®. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities and concerns.

© 2010 The College Board. College Board, ACCUPLACER, Advanced Placement Program, AP, AP Central, SAT, SpringBoard and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service is a trademark owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**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 Quick Reference found in the Appendix 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. An organization raises money by selling boxes of cookies. A cookie order specifies the variety of cookie and the number of boxes ordered. The declaration of the `CookieOrder` class is shown below.

```
public class CookieOrder
{
    /** Constructs a new CookieOrder object. */
    public CookieOrder(String variety, int numBoxes)
    { /* implementation not shown */ }

    /** @return the variety of cookie being ordered
     */
    public String getVariety()
    { /* implementation not shown */ }

    /** @return the number of boxes being ordered
     */
    public int getNumBoxes()
    { /* implementation not shown */ }

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

## 2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The MasterOrder class maintains a list of the cookies to be purchased. The declaration of the MasterOrder class is shown below.

```
public class MasterOrder
{
    /** The list of all cookie orders */
    private List<CookieOrder> orders;

    /** Constructs a new MasterOrder object. */
    public MasterOrder()
    {   orders = new ArrayList<CookieOrder>();   }

    /** Adds theOrder to the master order.
     *  @param theOrder the cookie order to add to the master order
     */
    public void addOrder(CookieOrder theOrder)
    {   orders.add(theOrder);   }

    /** @return the sum of the number of boxes of all of the cookie orders
     */
    public int getTotalBoxes()
    {   /* to be implemented in part (a) */   }

    /** Removes all cookie orders from the master order that have the same variety of
     *  cookie as cookieVar and returns the total number of boxes that were removed.
     *  @param cookieVar the variety of cookies to remove from the master order
     *  @return the total number of boxes of cookieVar in the cookie orders removed
     */
    public int removeVariety(String cookieVar)
    {   /* to be implemented in part (b) */   }

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

- (a) The getTotalBoxes method computes and returns the sum of the number of boxes of all cookie orders. If there are no cookie orders in the master order, the method returns 0.

Complete method getTotalBoxes below.

```
/** @return the sum of the number of boxes of all of the cookie orders
 */
public int getTotalBoxes()
```

## 2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) The `removeVariety` method updates the master order by removing all of the cookie orders in which the variety of cookie matches the parameter `cookieVar`. The master order may contain zero or more cookie orders with the same variety as `cookieVar`. The method returns the total number of boxes removed from the master order.

For example, consider the following code segment.

```
MasterOrder goodies = new MasterOrder();
goodies.addOrder(new CookieOrder("Chocolate Chip", 1));
goodies.addOrder(new CookieOrder("Shortbread", 5));
goodies.addOrder(new CookieOrder("Macaroon", 2));
goodies.addOrder(new CookieOrder("Chocolate Chip", 3));
```

After the code segment has executed, the contents of the master order are as shown in the following table.

"Chocolate Chip" 1	"Shortbread" 5	"Macaroon" 2	"Chocolate Chip" 3
-----------------------	-------------------	-----------------	-----------------------

The method call `goodies.removeVariety("Chocolate Chip")` returns 4 because there were two Chocolate Chip cookie orders totaling 4 boxes. The master order is modified as shown below.

"Shortbread" 5	"Macaroon" 2
-------------------	-----------------

The method call `goodies.removeVariety("Brownie")` returns 0 and does not change the master order.

Complete method `removeVariety` below.

```
/** Removes all cookie orders from the master order that have the same variety of
 * cookie as cookieVar and returns the total number of boxes that were removed.
 * @param cookieVar the variety of cookies to remove from the master order
 * @return the total number of boxes of cookieVar in the cookie orders removed
 */
public int removeVariety(String cookieVar)
```

## 2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. An `APLine` is a line defined by the equation  $ax + by + c = 0$ , where  $a$  is not equal to zero,  $b$  is not equal to zero, and  $a$ ,  $b$ , and  $c$  are all integers. The slope of an `APLine` is defined to be the `double` value  $-a/b$ . A point (represented by integers  $x$  and  $y$ ) is on an `APLine` if the equation of the `APLine` is satisfied when those  $x$  and  $y$  values are substituted into the equation. That is, a point represented by  $x$  and  $y$  is on the line if  $ax + by + c$  is equal to 0. Examples of two `APLine` equations are shown in the following table.

Equation	Slope ( $-a/b$ )	Is point (5, -2) on the line?
$5x + 4y - 17 = 0$	$-5/4 = -1.25$	Yes, because $5(5) + 4(-2) + (-17) = 0$
$-25x + 40y + 30 = 0$	$25/40 = 0.625$	No, because $-25(5) + 40(-2) + 30 \neq 0$

Assume that the following code segment appears in a class other than `APLine`. The code segment shows an example of using the `APLine` class to represent the two equations shown in the table.

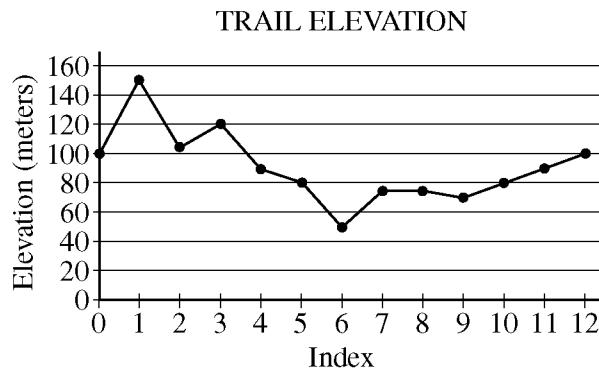
```
APLine line1 = new APLine(5, 4, -17);
double slope1 = line1.getSlope();           // slope1 is assigned -1.25
boolean onLine1 = line1.isOnLine(5, -2);    // true because 5(5) + 4(-2) + (-17) = 0

APLine line2 = new APLine(-25, 40, 30);
double slope2 = line2.getSlope();           // slope2 is assigned 0.625
boolean onLine2 = line2.isOnLine(5, -2);    // false because -25(5) + 40(-2) + 30 ≠ 0
```

Write the `APLine` class. Your implementation must include a constructor that has three integer parameters that represent  $a$ ,  $b$ , and  $c$ , in that order. You may assume that the values of the parameters representing  $a$  and  $b$  are not zero. It must also include a method `getSlope` that calculates and returns the slope of the line, and a method `isOnLine` that returns `true` if the point represented by its two parameters ( $x$  and  $y$ , in that order) is on the `APLine` and returns `false` otherwise. Your class must produce the indicated results when invoked by the code segment given above. You may ignore any issues related to integer overflow.

**2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

3. A hiking trail has elevation markers posted at regular intervals along the trail. Elevation information about a trail can be stored in an array, where each element in the array represents the elevation at a marker. The elevation at the first marker will be stored at array index 0, the elevation at the second marker will be stored at array index 1, and so forth. Elevations between markers are ignored in this question. The graph below shows an example of trail elevations.



The table below contains the data represented in the graph.

**Trail Elevation (meters)**

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Elevation	100	150	105	120	90	80	50	75	75	70	80	90	100

## 2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The declaration of the `Trail` class is shown below. You will write two unrelated methods of the `Trail` class.

```
public class Trail
{
    /** Representation of the trail. The number of markers on the trail is markers.length. */
    private int[] markers;

    /** Determines if a trail segment is level. A trail segment is defined by a starting marker,
     * an ending marker, and all markers between those two markers.
     * A trail segment is level if it has a difference between the maximum elevation
     * and minimum elevation that is less than or equal to 10 meters.
     * @param start the index of the starting marker
     * @param end the index of the ending marker
     * Precondition: 0 <= start < end <= markers.length - 1
     * @return true if the difference between the maximum and minimum
     *         elevation on this segment of the trail is less than or equal to 10 meters;
     *         false otherwise.
    */
    public boolean isLevelTrailSegment(int start, int end)
    { /* to be implemented in part (a) */ }

    /** Determines if this trail is rated difficult. A trail is rated by counting the number of changes in
     * elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
     * with 3 or more such changes is rated difficult.
     * @return true if the trail is rated difficult; false otherwise.
    */
    public boolean isDifficult()
    { /* to be implemented in part (b) */ }

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

- (a) Write the `Trail` method `isLevelTrailSegment`. A trail segment is defined by a starting marker, an ending marker, and all markers between those two markers. The parameters of the method are the index of the starting marker and the index of the ending marker. The method will return `true` if the difference between the maximum elevation and the minimum elevation in the trail segment is less than or equal to 10 meters.

For the trail shown at the beginning of the question, the trail segment starting at marker 7 and ending at marker 10 has elevations ranging between 70 and 80 meters. Because the difference between 80 and 70 is equal to 10, the trail segment is considered level.

The trail segment starting at marker 2 and ending at marker 12 has elevations ranging between 50 and 120 meters. Because the difference between 120 and 50 is greater than 10, this trail segment is not considered level.

## 2010 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

Complete method `isLevelTrailSegment` below.

```
/** Determines if a trail segment is level. A trail segment is defined by a starting marker,  
 * an ending marker, and all markers between those two markers.  
 * A trail segment is level if it has a difference between the maximum elevation  
 * and minimum elevation that is less than or equal to 10 meters.  
 * @param start the index of the starting marker  
 * @param end the index of the ending marker  
 * Precondition: 0 <= start < end <= markers.length - 1  
 * @return true if the difference between the maximum and minimum  
 * elevation on this segment of the trail is less than or equal to 10 meters;  
 * false otherwise.  
 */  
public boolean isLevelTrailSegment(int start, int end)
```

- (b) Write the `Trail` method `isDifficult`. A trail is rated by counting the number of changes in elevation that are at least 30 meters (up or down) between two consecutive markers. A trail with 3 or more such changes is rated difficult. The following table shows trail elevation data and the elevation changes between consecutive trail markers.

**Trail Elevation (meters)**

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Elevation	100	150	105	120	90	80	50	75	75	70	80	90	100
	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /
Elevation change	50	-45	15	-30	-10	-30	25	0	-5	10	10	10	10

This trail is rated difficult because it has 4 changes in elevation that are 30 meters or more (between markers 0 and 1, between markers 1 and 2, between markers 3 and 4, and between markers 5 and 6).

Complete method `isDifficult` below.

```
/** Determines if this trail is difficult. A trail is rated by counting the number of changes in  
 * elevation that are at least 30 meters (up or down) between two consecutive markers. A trail  
 * with 3 or more such changes is rated difficult.  
 * @return true if the trail is rated difficult; false otherwise.  
 */  
public boolean isDifficult()
```



## **AP® Computer Science A 2010 Scoring Guidelines**

### **The College Board**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the College Board is composed of more than 5,700 schools, colleges, universities and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,800 colleges through major programs and services in college readiness, college admission, guidance, assessment, financial aid and enrollment. Among its widely recognized programs are the SAT®, the PSAT/NMSQT®, the Advanced Placement Program® (AP®), SpringBoard® and ACCUPLACER®. The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities and concerns.

© 2010 The College Board. College Board, ACCUPLACER, Advanced Placement Program, AP, AP Central, SAT, SpringBoard and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service is a trademark owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

# AP® COMPUTER SCIENCE A

## 2010 GENERAL SCORING GUIDELINES

**Apply the question-specific rubric first.** To maintain scoring intent, a single error is generally accounted for only once per question thereby mitigating multiple penalties for the same error. The error categorization below is for cases not adequately covered by the question-specific rubric. Note that points can only be deducted if the error occurs in a part that has earned credit via the question-specific rubric. Any particular error is **penalized only once** in a question, even if it occurs on different parts of that question.

### Nonpenalized Errors

- spelling/case discrepancies if no ambiguity\*
- local variable not declared if others are declared in some part
- use keyword as identifier
- `[]` vs. `()` vs. `<>`
- = instead of == (and vice versa)
- length/size confusion for array, String, and ArrayList, with or without ()
- private qualifier on local variable
- extraneous code with no side effect; e.g., precondition check
- common mathematical symbols for operators ( $x \bullet \div \leq \geq < > \neq$ )
- missing {} where indentation clearly conveys intent and {} used elsewhere
- default constructor called without parens; e.g., new Fish;
- missing () on parameterless method call
- missing () around if/while conditions
- missing ; when majority are present
- missing public on class or constructor header
- extraneous [] when referencing entire array
- extraneous size in array declaration, e.g., `int[size] nums = new int[size];`

### Minor Errors (1/2 point)

- confused identifier (e.g., len for length or left() for getLeft())
- local variables used but none declared
- missing new in constructor call
- modifying a constant (final)
- use equals or compareTo method on primitives, e.g., `int x; ...x.equals(val)`
- array/collection access confusion ([] get)
- assignment dyslexia,  
e.g., `x + 3 = y; for y = x + 3;`
- `super(method())` instead of `super.method()`
- formal parameter syntax (with type) in method call, e.g., `a = method(int x)`
- missing public from method header when required
- "false"/"true" or 0/1 for boolean values
- "null" for null

### Major Errors (1 point)

- extraneous code that causes side effect; e.g., information written to output
- interface or class name instead of variable identifier; e.g., `Bug.move()` instead of `aBug.move()`
- `aMethod(obj)` instead of `obj.aMethod()`
- attempt to use private data or method when not accessible
- destruction of persistent data (e.g., changing value referenced by parameter)
- use class name in place of super in constructor or method call
- `void` method (or constructor) returns a value

#### Applying Minor Errors (½ point):

A minor error that occurs **exactly once** when the same concept is **correct two or more times** is regarded as an oversight and **not penalized**. A minor error **must be penalized** if it is the **only instance, one of two**, or occurs **two or more times**.

\* Spelling and case discrepancies for identifiers fall under the “nonpenalized” category only if the correction can be unambiguously inferred from context; for example, “ArrayList” instead of “ArrayList”. As a counter example, note that if a student declares “Bug bug;” then uses “Bug.move ()” instead of “bug.move ()”, the context does not allow for the reader to assume the object instead of the class.

# AP® COMPUTER SCIENCE A 2010 SCORING GUIDELINES

## Question 1: Master Order

<b>Part (a)</b>	<code>getTotalBoxes</code>	<b>3 points</b>
-----------------	----------------------------	-----------------

*Intent:* Compute and return the sum of the number of boxes of all cookie orders in `this.orders`

- +1 Considers all `CookieOrder` objects in `this.orders`
- +1/2 Accesses any element of `this.orders`
- +1/2 Accesses all elements of `this.orders` with no out-of-bounds access potential
  
- +1 1/2 Computes total number of boxes
  - +1/2 Creates an accumulator (declare and initialize)
  - +1/2 Invokes `getNumBoxes` on object of type `CookieOrder`
  - +1/2 Correctly accumulates total number of boxes
  
- +1/2 Returns computed total

<b>Part (b)</b>	<code>removeVariety</code>	<b>6 points</b>
-----------------	----------------------------	-----------------

*Intent:* Remove all `CookieOrder` objects from `this.orders` whose variety matches `cookieVar`; return total number of boxes removed

- +4 Identifies and removes matching `CookieOrder` objects
  - +1/2 Accesses an element of `this.orders`
  - +1/2 Compares parameter `cookieVar` with `getVariety()` of a `CookieOrder` object (must use `.equals` or `.compareTo`)
  - +1 Compares parameter `cookieVar` with `getVariety()` of all `CookieOrder` objects in `this.orders`, no out-of-bounds access potential
  - +1/2 Removes an element from `this.orders`
  - +1/2 Removes only matching `CookieOrder` objects
  - +1 Removes all matching `CookieOrder` objects, no elements skipped
  
- +1 1/2 Computes total number of boxes in removed `CookieOrder` objects
  - +1/2 Creates an accumulator (declare and initialize)
  - +1/2 Invokes `getNumBoxes` on object of type `CookieOrder`
  - +1/2 Correctly accumulates total number of boxes  
(must be in context of loop and match with `cookieVar`)
  
- +1/2 Returns computed total

### Usage:

- 1 consistently references incorrect name instead of `orders`, of potentially correct type
- 1 1/2 consistently references incorrect name instead of `orders`, incorrect type  
(e.g., `this`, `MasterOrder`)

# **AP<sup>®</sup> COMPUTER SCIENCE A 2010 SCORING GUIDELINES**

## **Question 2: APLine**

***Intent:*** Design complete APLine class including constructor, getSlope and isOnLine methods

**+1** Complete, correct header for APLine [ class APLine ]

*Note: Accept any visibility except private*

**+1 1/2** State maintenance

**+1/2** Declares at least one instance variable capable of maintaining numeric value

**+1/2** Declares at least three instance variables capable of maintaining numeric values

**+1/2** All state variables have private visibility

*Note: Accept any numeric type (primitive or object)*

*Note: Accept any distinct Java-valid variable names*

**+1 1/2** APLine Constructor

*Method header*

**+1/2** Correctly formed header (visibility not private; name APLine)

**+1/2** Specifies exactly three numeric parameters

*Method body*

**+1/2** Sets appropriate state variables based on parameters (no shadowing errors)

*Note: Interpret instance fields by usage not by name*

**+2 1/2** getSlope

*Method header*

**+1/2** Correct method header

(visibility not private; type double or Double; name getSlope; parameterless)

*Method body*

**+1/2** Computation uses correct formula for slope

**+1** Computation uses double precision (no integer division)

**+1/2** Returns computed value

**+2 1/2** isOnLine

*Method header*

**+1/2** Correct formed header (visibility not private; type boolean or Boolean, name isOnLine)

**+1/2** Specifies exactly two numeric parameters

*Method body*

**+1/2** Computation uses correct formula involving state and parameters  
$$(a*x + b*y + c)$$

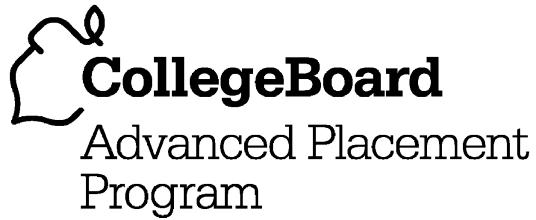
**+1/2** Computation uses correct comparison test (equal to zero)

**+1/2** Returns true if is on this APLine; false otherwise

# AP® COMPUTER SCIENCE A 2010 SCORING GUIDELINES

## Question 3: Trail

Part (a)	isLevelTrailSegment	5 points
<i>Intent:</i> Return true if maximum difference $\leq 10$ (segment is level); false otherwise		
<p>+3 Determination of information needed to test level-trail condition</p> <p>+1/2 Creates and maintains local state for determination of maximum (or minimum); <i>alternate solution:</i> tests difference in elevations</p> <p>+1/2 Accesses the value of any element of <code>this.markers</code></p> <p>+1 All and only appropriate elements of <code>this.markers</code> participate in determination of information needed to test level-trail condition; no out-of-bounds access potential</p> <p>+1 Compares element to state in context of updating maximum (or minimum); <i>alternate solution:</i> tests difference in elevations</p> <p>+1 Correctly determines information needed to test level-trail condition for the elements examined; must address two or more pairs of elements</p> <p>+1 Returns true if determined maximum difference is <math>\leq 10</math>, false otherwise</p>		
Part (b)	isDifficult	4 points
<i>Intent:</i> Return true if trail is difficult (based on number of changes of given magnitude); false otherwise		
<p>+3 Determine number of changes, greater than or equal to 30, between consecutive values in <code>this.markers</code></p> <p>+1/2 Creates, initializes and accumulates a count of number of changes</p> <p>+1/2 Accesses the value of any element of <code>this.markers</code> in context of iteration</p> <p>+1/2 Accesses the value of all elements of <code>this.markers</code>, no out-of-bounds access potential</p> <p>+1/2 Computes difference of all and only consecutive values in <code>this.markers</code></p> <p>+1 Updates accumulated count if and only if absolute value of difference is <math>\geq 30</math></p> <p>+1 Returns true if accumulated count is <math>\geq 3</math>; false otherwise</p>		



## **AP® Computer Science A 2009 Free-Response Questions**

### **The College Board**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,600 schools, colleges, universities and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools and 3,800 colleges through major programs and services in college readiness, college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT® and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities and concerns.

© 2009 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.

Permission to use copyrighted College Board materials may be requested online at:  
[www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**COMPUTER SCIENCE A  
SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

**Notes:**

- Assume that the classes listed in the Quick Reference found in the Appendix 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. A statistician is studying sequences of numbers obtained by repeatedly tossing a six-sided number cube. On each side of the number cube is a single number in the range of 1 to 6, inclusive, and no number is repeated on the cube. The statistician is particularly interested in runs of numbers. A run occurs when two or more consecutive tosses of the cube produce the same value. For example, in the following sequence of cube tosses, there are runs starting at positions 1, 6, 12, and 14.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	5	5	4	3	1	2	2	2	2	6	1	3	3	5	5	5	5

The number cube is represented by the following class.

```
public class NumberCube
{
    /** @return an integer value between 1 and 6, inclusive
     */
    public int toss()
    { /* implementation not shown */ }

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

You will implement a method that collects the results of several tosses of a number cube and another method that calculates the longest run found in a sequence of tosses.

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the method `getCubeTosses` that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times. Complete method `getCubeTosses` below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.  
 * @param cube a NumberCube  
 * @param numTosses the number of tosses to be recorded  
 *      Precondition: numTosses > 0  
 * @return an array of numTosses values  
 */  
public static int[] getCubeTosses(NumberCube cube, int numTosses)
```

- (b) Write the method `getLongestRun` that takes as its parameter an array of integer values representing a series of number cube tosses. The method returns the starting index in the array of a run of maximum size. A run is defined as the repeated occurrence of the same value in two or more consecutive positions in the array.

For example, the following array contains two runs of length 4, one starting at index 6 and another starting at index 14. The method may return either of those starting indexes.

If there are no runs of any value, the method returns `-1`.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	5	5	4	3	1	2	2	2	2	6	1	3	3	5	5	5	5

Complete method `getLongestRun` below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values  
 * in the array values.  
 * @param values an array of integer values representing a series of number cube tosses  
 *      Precondition: values.length > 0  
 * @return the starting index of a run of maximum size;  
 *         -1 if there is no run  
 */  
public static int getLongestRun(int[] values)
```

**2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

3. An electric car that runs on batteries must be periodically recharged for a certain number of hours. The battery technology in the car requires that the charge time not be interrupted.

The cost for charging is based on the hour(s) during which the charging occurs. A rate table lists the 24 one-hour periods, numbered from 0 to 23, and the corresponding hourly cost for each period. The same rate table is used for each day. Each hourly cost is a positive integer. A sample rate table is given below.

Hour	Cost
0	50
1	60
2	160
3	60
4	80
5	100
6	100
7	120

Hour	Cost
8	150
9	150
10	150
11	200
12	40
13	240
14	220
15	220

Hour	Cost
16	200
17	200
18	180
19	180
20	140
21	100
22	80
23	60

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The class `BatteryCharger` below uses a rate table to determine the most economic time to charge the battery. You will write two of the methods for the `BatteryCharger` class.

```
public class BatteryCharger
{
    /** rateTable has 24 entries representing the charging costs for hours 0 through 23. */
    private int[] rateTable;

    /** Determines the total cost to charge the battery starting at the beginning of startHour.
     * @param startHour the hour at which the charge period begins
     *
     * Precondition:  $0 \leq \text{startHour} \leq 23$ 
     * @param chargeTime the number of hours the battery needs to be charged
     *
     * Precondition:  $\text{chargeTime} > 0$ 
     * @return the total cost to charge the battery
     */
    private int getChargingCost(int startHour, int chargeTime)
    { /* to be implemented in part (a) */ }

    /** Determines start time to charge the battery at the lowest cost for the given charge time.
     * @param chargeTime the number of hours the battery needs to be charged
     *
     * Precondition:  $\text{chargeTime} > 0$ 
     * @return an optimal start time, with  $0 \leq \text{returned value} \leq 23$ 
     */
    public int getChargeStartTime(int chargeTime)
    { /* to be implemented in part (b) */ }

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

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.  
 * @param startHour the hour at which the charge period begins  
 * Precondition:  $0 \leq \text{startHour} \leq 23$   
 * @param chargeTime the number of hours the battery needs to be charged  
 * Precondition:  $\text{chargeTime} > 0$   
 * @return the total cost to charge the battery  
 */  
private int getChargingCost(int startHour, int chargeTime)
```

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the `BatteryCharger` method `getChargeStartTime` that returns the start time that will allow the battery to be charged at minimal cost. If there is more than one possible start time that produces the minimal cost, any of those start times can be returned.

For example, using the rate table given at the beginning of the question, the following table shows the resulting minimal costs and optimal starting hour of several possible charges.

Hours of Charge Time	Minimum Cost	Start Hour of Charge	Last Hour of Charge
1	40	12	12
2	110	0	1
		23	0 (the next day)
7	550	22	4 (the next day)
30	3,710	22	3 (two days later)

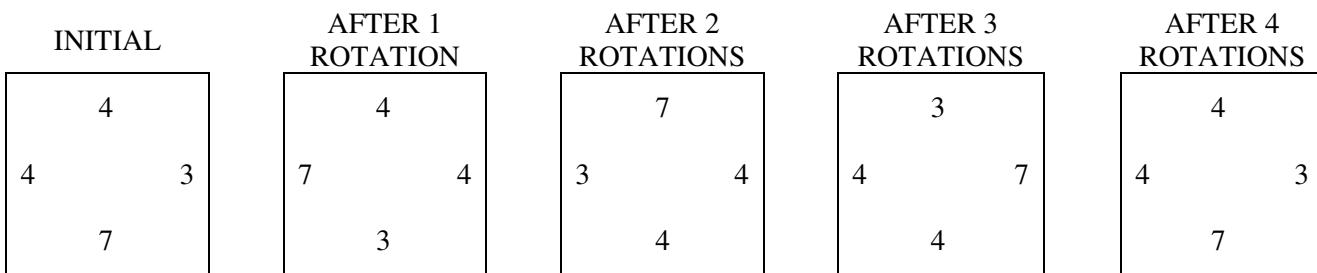
Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.  
 * @param chargeTime the number of hours the battery needs to be charged  
 * Precondition: chargeTime > 0  
 * @return an optimal start time, with 0 ≤ returned value ≤ 23  
 */  
public int getChargeStartTime(int chargeTime)
```

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. A game uses square tiles that have numbers on their sides. Each tile is labeled with a number on each of its four sides and may be rotated clockwise, as illustrated below.



The tiles are represented by the `NumberTile` class, as given below.

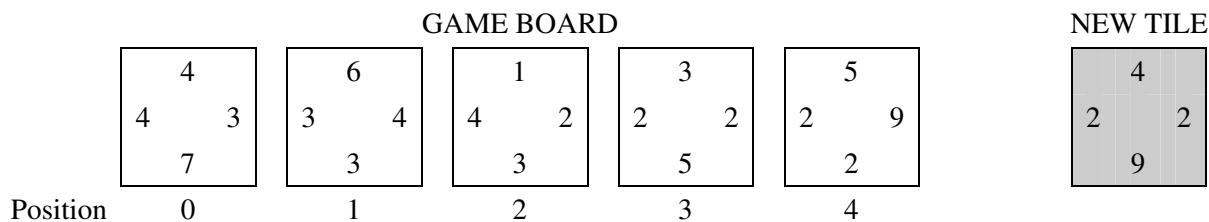
```
public class NumberTile
{
    /** Rotates the tile 90 degrees clockwise
     */
    public void rotate()
    { /* implementation not shown */ }

    /** @return value at left edge of tile
     */
    public int getLeft()
    { /* implementation not shown */ }

    /** @return value at right edge of tile
     */
    public int getRight()
    { /* implementation not shown */ }

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

Tiles are placed on a game board so that the adjoining sides of adjacent tiles have the same number. The following figure illustrates an arrangement of tiles and shows a new tile that is to be placed on the game board.



## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

In its original orientation, the new tile can be inserted between the tiles at positions 2 and 3 or between the tiles at positions 3 and 4. If the new tile is rotated once, it can be inserted before the tile at position 0 (the first tile) or after the tile at position 4 (the last tile). Assume that the new tile, in its original orientation, is inserted between the tiles at positions 2 and 3. As a result of the insertion, the tiles at positions 3 and 4 are moved one location to the right, and the new tile is inserted at position 3, as shown below.

GAME BOARD AFTER INSERTING TILE							
Position	0	1	2	3	4	5	
	4 4 7	6 3 3	1 4 3	4 2 3	2 2 9	3 2 5	5 2 2

A partial definition of the `TileGame` class is given below.

```
public class TileGame
{
    /** represents the game board; guaranteed never to be null */
    private ArrayList<NumberTile> board;

    public TileGame()
    { board = new ArrayList<NumberTile>(); }

    /** Determines where to insert tile, in its current orientation, into game board
     *  @param tile the tile to be placed on the game board
     *  @return the position of tile where tile is to be inserted:
     *          0 if the board is empty;
     *          -1 if tile does not fit in front, at end, or between any existing tiles;
     *          otherwise, 0 ≤ position returned ≤ board.size()
     */
    private int getIndexForFit(NumberTile tile)
    { /* to be implemented in part (a) */ }

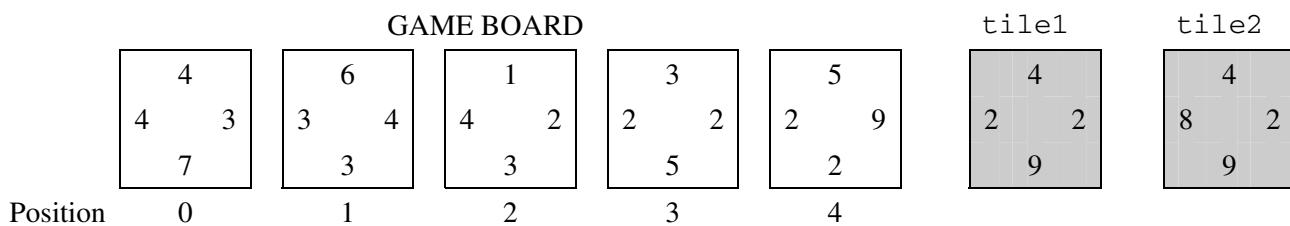
    /** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
     *  If there are no tiles on the game board, the tile is placed at position 0.
     *  The tile should be placed at most 1 time.
     *  Precondition: board is not null
     *  @param tile the tile to be placed on the game board
     *  @return true if tile is placed successfully; false otherwise
     *  Postcondition: the orientations of the other tiles on the board are not changed
     *  Postcondition: the order of the other tiles on the board relative to each other is not changed
     */
    public boolean insertTile(NumberTile tile)
    { /* to be implemented in part (b) */ }

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

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write the TileGame method `getIndexForFit` that determines where a given tile, in its current orientation, fits on the game board. A tile can be inserted at either end of a game board or between two existing tiles if the side(s) of the new tile match the adjacent side(s) of the tile(s) currently on the game board. If there are no tiles on the game board, the position for the insert is 0. The method returns the position that the new tile will occupy on the game board after it has been inserted. If there are multiple possible positions for the tile, the method will return any one of them. If the given tile does not fit anywhere on the game board, the method returns -1.

For example, the following diagram shows a game board and two potential tiles to be placed. The call `getIndexForFit(tile1)` can return either 3 or 4 because `tile1` can be inserted between the tiles at positions 2 and 3, or between the tiles at positions 3 and 4. The call `getIndexForFit(tile2)` returns -1 because `tile2`, in its current orientation, does not fit anywhere on the game board.



Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 * @param tile the tile to be placed on the game board
 * @return the position of tile where tile is to be inserted:
 *         0 if the board is empty;
 *         -1 if tile does not fit in front, at end, or between any existing tiles;
 *         otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
```

## 2009 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the TileGame method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

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

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 * If there are no tiles on the game board, the tile is placed at position 0.
 * The tile should be placed at most 1 time.
 * Precondition: board is not null
 * @param tile the tile to be placed on the game board
 * @return true if tile is placed successfully; false otherwise
 * Postcondition: the orientations of the other tiles on the board are not changed
 * Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
```

**STOP**

**END OF EXAM**



## **AP® Computer Science A 2009 Scoring Guidelines**

### **The College Board**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,600 schools, colleges, universities and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools and 3,800 colleges through major programs and services in college readiness, college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT® and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities and concerns.

© 2009 The College Board. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.

Permission to use copyrighted College Board materials may be requested online at:  
[www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

**Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central® is the official online home for AP teachers: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).**

**AP® COMPUTER SCIENCE A  
2009 SCORING GUIDELINES**

**Question 1: Number Cube**

<b>Part (a)</b>	<code>getCubeTosses</code>	<b>4 points</b>
-----------------	----------------------------	-----------------

- +1 constructs array
  - +1/2 constructs an array of type `int` **or** size `numTosses`
  - +1/2 constructs an array of type `int` **and** size `numTosses`
- +2 1/2 processes tosses
  - +1 repeats execution of statements `numTosses` times
  - +1 tosses cube in context of iteration
  - +1/2 collects results of tosses
- +1/2 returns array of generated results

<b>Part (b)</b>	<code>getLongestRun</code>	<b>5 points</b>
-----------------	----------------------------	-----------------

- +1 iterates over `values`
  - +1/2 accesses element of `values` in context of iteration
  - +1/2 accesses all elements of `values`, no out-of-bounds access potential
- +1 determines existence of run of consecutive elements
  - +1/2 comparison involving an element of `values`
  - +1/2 comparison of consecutive elements of `values`
- +1 always determines length of at least one run of consecutive elements
- +1 identifies maximum length run based on all runs
- +1 return value
  - +1/2 returns starting index of identified maximum length run
  - +1/2 returns -1 if no run identified

**AP® COMPUTER SCIENCE A  
2009 SCORING GUIDELINES**

**Question 3: Battery Charger**

<b>Part (a)</b>	getChargingCost	<b>5 points</b>
-----------------	-----------------	-----------------

- +1 1/2 accesses array elements
  - +1/2 accesses any element of `rateTable`
  - +1/2 accesses an element of `rateTable` using an index derived from `startHour`
  - +1/2 accesses multiple elements of `rateTable` with no out-of-bounds access potential
  
- +2 1/2 accumulates values
  - +1/2 declares and initializes an accumulator
  - +1/2 accumulates values from elements of `rateTable`
  - +1/2 selects values from `rateTable` using an index derived from `startHour` and `chargeTime`
  - +1 determines correct sum of values from `rateTable` based on `startHour` and `chargeTime`
  
- +1 value returned
  - +1/2 returns any nonconstant (derived) value
  - +1/2 returns accumulated value

<b>Part (b)</b>	getChargeStartTime	<b>4 points</b>
-----------------	--------------------	-----------------

- +1/2 invokes `getChargingCost` or replicates functionality with no errors
  
- +1 determines charging cost
  - +1/2 considers **all** potential start times; must include at least 0 ... 23
  - +1/2 determines charging cost for potential start times

*Note: No penalty here for parameter passed to `getChargingCost` that violates its preconditions (e.g., 24)*
  
- +1 compares charging costs for two different start times
  
- +1 determines minimum charging cost based on potential start times

*Note: Penalty here for using result of call to `getChargingCost` that violates its preconditions (e.g., 24)*
  
- +1/2 returns start time for minimum charging cost

**AP® COMPUTER SCIENCE A  
2009 SCORING GUIDELINES**

**Question 4: Tile Game**

<b>Part (a)</b>	<code>getIndexForFit</code>	<b>6 points</b>
-----------------	-----------------------------	-----------------

- +1 empty board
  - +1/2 checks for zero-sized board
  - +1/2 returns 0 if empty board detected
- +1 accesses tiles from board
  - +1/2 accesses any tile from board
  - +1/2 accesses all tiles of board (as appropriate) with no out-of-bounds access potential
- +1 uses tile values
  - +1/2 accesses left or right value of any tile
  - +1/2 compares left (right) value of parameter with right (left) value of any tile from board
- +2 determines tile fit
  - +1/2 only right value of parameter compared with left value of initial tile of board
  - +1/2 only left value of parameter compared with right value of final tile of board
  - +1 compares appropriate values of parameter and interior tiles of board
- +1 result
  - +1/2 returns located index if tile fits in board
  - +1/2 returns -1 if tile does not fit in board

<b>Part (b)</b>	<code>insertTile</code>	<b>3 points</b>
-----------------	-------------------------	-----------------

- +1/2 invokes `getIndexForFit` or replicates functionality with no errors
- +1 1/2 tile orientation
  - +1/2 invokes `rotate` on parameter
  - +1/2 performs **all** necessary rotations
  - +1/2 invokes `getIndexForFit` for each necessary orientation
- +1/2 adds tile correctly and only if `getIndexForFit` returns value other than -1
- +1/2 returns `true` if `getIndexForFit` returns value other than -1; `false` otherwise



## **AP® Computer Science A 2008 Free-Response Questions**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2008 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.

Permission to use copyrighted College Board materials may be requested online at:  
[www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

**Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).**

**AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).**

# **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

## **COMPUTER SCIENCE A SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

**Notes:**

- Assume that the classes listed in the Quick Reference found in the Appendix 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. A travel agency maintains a list of information about airline flights. Flight information includes a departure time and an arrival time. You may assume that the two times occur on the same day. These times are represented by objects of the `Time` class.

The declaration for the `Time` class is shown below. It includes a method `minutesUntil` that returns the difference (in minutes) between the current `Time` object and another `Time` object.

```
public class Time
{
    /**
     * @return difference, in minutes, between this time and other;
     *         difference is negative if other is earlier than this time
     */
    public int minutesUntil(Time other)
    { /* implementation not shown */ }

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

For example, assume that `t1` and `t2` are `Time` objects where `t1` represents 1:00 P.M. and `t2` represents 2:15 P.M. The call `t1.minutesUntil(t2)` will return 75 and the call `t2.minutesUntil(t1)` will return -75.

## **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

The declaration for the `Flight` class is shown below. It has methods to access the departure time and the arrival time of a flight. You may assume that the departure time of a flight is earlier than its arrival time.

```
public class Flight
{
    /** @return time at which the flight departs
     */
    public Time getDepartureTime()
    { /* implementation not shown */ }

    /** @return time at which the flight arrives
     */
    public Time getArrivalTime()
    { /* implementation not shown */ }

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

## 2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

A trip consists of a sequence of flights and is represented by the `Trip` class. The `Trip` class contains an `ArrayList` of `Flight` objects that are stored in chronological order. You may assume that for each flight after the first flight in the list, the departure time of the flight is later than the arrival time of the preceding flight in the list. A partial declaration of the `Trip` class is shown below. You will write two methods for the `Trip` class.

```
public class Trip
{
    private ArrayList<Flight> flights;
    // stores the flights (if any) in chronological order

    /** @return the number of minutes from the departure of the first flight to the arrival
     *          of the last flight if there are one or more flights in the trip;
     *          0, if there are no flights in the trip
     */
    public int getDuration()
    { /* to be implemented in part (a) */ }

    /** Precondition: the departure time for each flight is later than the arrival time of its
     *          preceding flight
     * @return the smallest number of minutes between the arrival of a flight and the departure
     *          of the flight immediately after it, if there are two or more flights in the trip;
     *          -1, if there are fewer than two flights in the trip
     */
    public int getShortestLayover()
    { /* to be implemented in part (b) */ }

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

## 2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

(a) Complete method `getDuration` below.

```
/** @return the number of minutes from the departure of the first flight to the arrival  
 *         of the last flight if there are one or more flights in the trip;  
 *         0, if there are no flights in the trip  
 */  
public int getDuration()
```

(b) Write the `Trip` method `getShortestLayover`. A layover is the number of minutes from the arrival of one flight in a trip to the departure of the flight immediately after it. If there are two or more flights in the trip, the method should return the shortest layover of the trip; otherwise, it should return -1.

For example, assume that the instance variable `flights` of a `Trip` object `vacation` contains the following flight information.

	<b>Departure Time</b>	<b>Arrival Time</b>	<b>Layover (minutes)</b>
Flight 0	11:30 a.m.	12:15 p.m.	
Flight 1	1:15 p.m.	3:45 p.m.	} 60
Flight 2	4:00 p.m.	6:45 p.m.	} 15
Flight 3	10:15 p.m.	11:00 p.m.	} 210

The call `vacation.getShortestLayover()` should return 15.

Complete method `getShortestLayover` below.

```
/** Precondition: the departure time for each flight is later than the arrival time of its  
 *         preceding flight  
 * @return the smallest number of minutes between the arrival of a flight and the departure  
 *         of the flight immediately after it, if there are two or more flights in the trip;  
 *         -1, if there are fewer than two flights in the trip  
 */  
public int getShortestLayover()
```

## 2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. Consider a method of encoding and decoding words that is based on a *master string*. This master string will contain all the letters of the alphabet, some possibly more than once. An example of a master string is "sixtyzipperswerequicklypickedfromthewovenjutebag". This string and its indexes are shown below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	i	x	t	y	z	i	p	p	e	r	s	w	e	r	e	q	u	i	c	k	l	y	p
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
i	c	k	e	d	f	r	o	m	t	h	e	w	o	v	e	n	j	u	t	e	b	a	g

An encoded string is defined by a list of *string parts*. A string part is defined by its starting index in the master string and its length. For example, the string "overeager" is encoded as the list of string parts [ (37, 3), (14, 2), (46, 2), (9, 2) ] denoting the substrings "ove", "re", "ag", and "er".

String parts will be represented by the `StringPart` class shown below.

```
public class StringPart
{
    /** @param start the starting position of the substring in a master string
     *  @param length the length of the substring in a master string
     */
    public StringPart(int start, int length)
    { /* implementation not shown */ }

    /** @return the starting position of the substring in a master string
     */
    public int getStart()
    { /* implementation not shown */ }

    /** @return the length of the substring in a master string
     */
    public int getLength()
    { /* implementation not shown */ }

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

## 2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The class `StringCoder` provides methods to encode and decode words using a given master string. When encoding, there may be multiple matching string parts of the master string. The helper method `findPart` is provided to choose a string part within the master string that matches the beginning of a given string.

```
public class StringCoder
{
    private String masterString;

    /** @param master the master string for the StringCoder
     *      Precondition: the master string contains all the letters of the alphabet
     */
    public StringCoder(String master)
    {   masterString = master;   }

    /** @param parts an ArrayList of string parts that are valid in the master string
     *      Precondition: parts.size() > 0
     *      @return the string obtained by concatenating the parts of the master string
     */
    public String decodeString(ArrayList<StringPart> parts)
    {   /* to be implemented in part (a) */   }

    /** @param str the string to encode using the master string
     *      Precondition: all of the characters in str appear in the master string;
     *                  str.length() > 0
     *      @return a string part in the master string that matches the beginning of str.
     *              The returned string part has length at least 1.
     */
    private StringPart findPart(String str)
    {   /* implementation not shown */   }

    /** @param word the string to be encoded
     *      Precondition: all of the characters in word appear in the master string;
     *                  word.length() > 0
     *      @return an ArrayList of string parts of the master string that can be combined
     *              to create word
     */
    public ArrayList<StringPart> encodeString(String word)
    {   /* to be implemented in part (b) */   }

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

## **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the `StringCoder` method `decodeString`. This method retrieves the substrings in the master string represented by each of the `StringPart` objects in `parts`, concatenates them in the order in which they appear in `parts`, and returns the result.

Complete method `decodeString` below.

```
/** @param parts an ArrayList of string parts that are valid in the master string
 *      Precondition: parts.size() > 0
 *      @return the string obtained by concatenating the parts of the master string
 */
public String decodeString(ArrayList<StringPart> parts)
```

## **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) Write the `StringCoder` method `encodeString`. A string is encoded by determining the substrings in the master string that can be combined to generate the given string. The encoding starts with a string part that matches the beginning of the word, followed by a string part that matches the beginning of the rest of the word, and so on. The string parts are returned in an array list in the order in which they appear in `word`.

The helper method `findPart` must be used to choose matching string parts in the master string.

Complete method `encodeString` below.

```
/** @param word the string to be encoded
 *
 *      Precondition: all of the characters in word appear in the master string;
 *      word.length() > 0
 *
 *      @return an ArrayList of string parts of the master string that can be combined
 *              to create word
 */
public ArrayList<StringPart> encodeString(String word)
```

## 2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. A *checker* is an object that examines strings and *accepts* those strings that meet a particular criterion.

The Checker interface is defined below.

```
public interface Checker
{
    /** @param text a string to consider for acceptance
     *  @return true if this Checker accepts text; false otherwise
     */
    boolean accept(String text);
}
```

In this question, you will write two classes that implement the Checker interface. You will then create a Checker object that checks for a particular acceptance criterion.

- (a) A SubstringChecker accepts any string that contains a particular substring. For example, the following SubstringChecker object broccoliChecker accepts all strings containing the substring "broccoli".

```
Checker broccoliChecker = new SubstringChecker("broccoli");
```

The following table illustrates the results of several calls to the broccoliChecker accept method.

Method Call	Result
broccoliChecker.accept("broccoli")	true
broccoliChecker.accept("I like broccoli")	true
broccoliChecker.accept("carrots are great")	false
broccoliChecker.accept("Broccoli Bonanza")	false

Write the SubstringChecker class that implements the Checker interface. The constructor should take a single String parameter that represents the particular substring to be matched.

## **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) Checkers can be created to check for multiple acceptance criteria by combining other checker objects. For example, an `AndChecker` is a `Checker` that is constructed with two objects of classes that implement the `Checker` interface (such as `SubstringChecker` or `AndChecker` objects). The `AndChecker` `accept` method returns `true` if and only if the string is accepted by both of the `Checker` objects with which it was constructed.

In the code segment below, the `bothChecker` object accepts all strings containing both "beets" and "carrots". The code segment also shows how the `veggies` object can be constructed to accept all strings containing the three substrings "beets", "carrots", and "artichokes".

```
Checker bChecker = new SubstringChecker("beets");
Checker cChecker = new SubstringChecker("carrots");
Checker bothChecker = new AndChecker(bChecker, cChecker);

Checker aChecker = new SubstringChecker("artichokes");
Checker veggies = new AndChecker(bothChecker, aChecker);
```

The following table illustrates the results of several calls to the `bothChecker` `accept` method and the `veggies` `accept` method.

Method Call	Result
<code>bothChecker.accept("I love beets and carrots")</code>	<code>true</code>
<code>bothChecker.accept("beets are great")</code>	<code>false</code>
<code>veggies.accept("artichokes, beets, and carrots")</code>	<code>true</code>

Write the `AndChecker` class that implements the `Checker` interface. The constructor should take two `Checker` parameters.

## **2008 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

(c) Another implementation of the `Checker` interface is the `NotChecker`, which contains the following:

- A one-parameter constructor that takes one `Checker` object
- An `accept` method that returns `true` if and only if its `Checker` object does NOT accept the string

Using any of the classes `SubstringChecker`, `AndChecker`, and `NotChecker`, construct a `Checker` that accepts a string if and only if it contains neither the substring "artichokes" nor the substring "kale". Assign the constructed `Checker` to `yummyChecker`. Consider the following incomplete code segment.

```
Checker aChecker = new SubstringChecker("artichokes");
Checker kChecker = new SubstringChecker("kale");
Checker yummyChecker;
/* code to construct and assign to yummyChecker */
```

The following table illustrates the results of several calls to the `yummyChecker accept` method.

Method Call	Result
<code>yummyChecker.accept("chocolate truffles")</code>	<code>true</code>
<code>yummyChecker.accept("kale is great")</code>	<code>false</code>
<code>yummyChecker.accept("Yuck: artichokes &amp; kale")</code>	<code>false</code>

In writing your solution, you may use any of the classes specified for this problem. Assume that these classes work as specified, regardless of what you wrote in parts (a) and (b). You may assume that the declarations for `aChecker`, `kChecker`, and `yummyChecker` in the code segment above have already been executed.

Write your `/* code to construct and assign to yummyChecker */` below.

**STOP**

**END OF EXAM**



## **AP® Computer Science A 2008 Scoring Guidelines**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,400 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2008 The College Board. All rights reserved. College Board, AP Central, Advanced Placement Program, AP, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central is the online home for AP teachers: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**AP® COMPUTER SCIENCE A  
2008 SCORING GUIDELINES**

**Question 1: Flight List**

<b>Part A:</b>	<b>getDuration</b>	<b>4 points</b>
----------------	--------------------	-----------------

- +1 handle empty case
  - +1/2 check if `flights` is empty
  - +1/2 return 0 if empty
- +1 access start time
  - +1/2 access `flights.get(0)`
  - +1/2 correctly call `getDepartureTime` on a flight
- +1 access end time
  - +1/2 access `flights.get(flights.size() - 1)`
  - +1/2 correctly call `getArrivalTime` on a flight
- +1 calculate and return duration
  - +1/2 call `minutesUntil` using `Time` objects
  - +1/2 return correct duration (using `minutesUntil`)

<b>Part B:</b>	<b>getShortestLayover</b>	<b>5 points</b>
----------------	---------------------------	-----------------

- +1 handle case with 0 or 1 flight
  - +1/2 check if `flights.size() < 2`
  - +1/2 return -1 in that case
- +1 traverse `flights`
  - +1/2 correctly access an element of `flights` (in context of loop)
  - +1/2 access all elements of `flights` (lose this if index out-of-bounds)
- +2 1/2 find shortest layover (in context of loop)
  - +1 get layover time between successive flights (using `minutesUntil`)
  - +1/2 compare layover time with some previous layover
  - +1 correctly identify shortest layover
- +1/2 return shortest layover

**AP® COMPUTER SCIENCE A  
2008 SCORING GUIDELINES**

**Question 2: String Coder**

<b>Part A:</b>	decodeString	<b>4 1/2 points</b>
----------------	--------------	---------------------

- +1 traverse parts
- +1/2 correctly access an element of parts (in context of loop)
- +1/2 access all elements of parts (lose this if index out-of-bounds)
  
- +2 retrieve substrings from masterString
- +1/2 correctly call `getStart()` and `getLength()` on accessed part
- +1 1/2 extract a substring from masterString
  - +1/2 `masterString.substring(X, Y)`
  - +1 extract correct substring
  
- +1 1/2 build and return decoded string
  - +1 correctly build string from substrings of masterString
  - +1/2 return built string

<b>Part B:</b>	encodeString	<b>4 1/2 points</b>
----------------	--------------	---------------------

- +1/2 construct an `ArrayList<StringPart>` (must assign to a variable, generic okay)
  
- +3 1/2 find, collect string parts, and build list (in context of loop)
  - +1 `findPart(X)`, where X is word or a substring of word
  - +1 calls to `findPart` involve progressively smaller suffixes of word
  - +1/2 add found string part to `ArrayList` of string parts
  - +1 build correct list of string parts (must have used `findPart`)
  
- +1/2 return `ArrayList` of string parts

**AP® COMPUTER SCIENCE A  
2008 SCORING GUIDELINES**

**Question 4: Checker Objects (Design)**

<b>Part A:</b>	SubstringChecker	<b>4 points</b>
----------------	------------------	-----------------

- +1/2 class SubstringChecker implements Checker
- +1/2 declare private instance variable of type String
- +1 constructor
  - +1/2 SubstringChecker(String *goalString*)
  - +1/2 initialize instance variable to parameter
- +2 accept method
  - +1/2 public boolean accept(String *text*)
  - +1 1/2 determine whether to accept
    - +1/2 attempt to find instance variable in *text*  
(either call indexOf, contains, or compare with substrings)
    - +1 return correct boolean value in all cases

<b>Part B:</b>	AndChecker	<b>4 points</b>
----------------	------------	-----------------

- +1/2 class AndChecker implements Checker
- +1/2 declare private instance variable(s) capable of storing two Checker objects
- +1 constructor
  - +1/2 AndChecker(Checker *c1*, Checker *c2*)
  - +1/2 initialize instance variable(s) to parameters
- +2 accept method
  - +1/2 public boolean accept(String *text*)
  - +1 1/2 determine whether to accept
    - +1/2 attempt to call accept(*text*) on both stored Checkers
    - +1 return correct boolean value in all cases

<b>Part C:</b>	yummyChecker	<b>1 point</b>
----------------	--------------	----------------

- +1 correctly assign yummyChecker



## **AP® Computer Science A 2007 Free-Response Questions**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2007 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.

Permission to use copyrighted College Board materials may be requested online at:  
[www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

# **2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

## **COMPUTER SCIENCE A SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix 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. A positive integer is called a "self-divisor" if every decimal digit of the number is a divisor of the number, that is, the number is evenly divisible by each and every one of its digits. For example, the number 128 is a self-divisor because it is evenly divisible by 1, 2, and 8. However, 26 is not a self-divisor because it is not evenly divisible by the digit 6. Note that 0 is not considered to be a divisor of any number, so any number containing a 0 digit is NOT a self-divisor. There are infinitely many self-divisors.

```
public class SelfDivisor
{
    /** @param number the number to be tested
     *  Precondition: number > 0
     *  @return true if every decimal digit of number is a divisor of number;
     *          false otherwise
     */
    public static boolean isSelfDivisor(int number)
    { /* to be implemented in part (a) */ }

    /** @param start starting point for values to be checked
     *  Precondition: start > 0
     *  @param num the size of the array to be returned
     *  Precondition: num > 0
     *  @return an array containing the first num integers  $\geq$  start that are self-divisors
     */
    public static int[] firstNumSelfDivisors(int start, int num)
    { /* to be implemented in part (b) */ }

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

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Write method `isSelfDivisor`, which takes a positive integer as its parameter. This method returns `true` if the number is a self-divisor; otherwise, it returns `false`.

Complete method `isSelfDivisor` below.

```
/** @param number the number to be tested
 *      Precondition: number > 0
 *      @return true if every decimal digit of number is a divisor of number;
 *              false otherwise
 */
public static boolean isSelfDivisor(int number)
```

- (b) Write method `firstNumSelfDivisors`, which takes two positive integers as parameters, representing a start value and a number of values. Method `firstNumSelfDivisors` returns an array of size `num` that contains the first `num` self-divisors that are greater than or equal to `start`.

For example, the call `firstNumSelfDivisors(10, 3)` should return an array containing the values 11, 12, and 15, because the first three self-divisors that are greater than or equal to 10 are 11, 12, and 15.

In writing `firstNumSelfDivisors`, assume that `isSelfDivisor` works as specified, regardless of what you wrote in part (a).

Complete method `firstNumSelfDivisors` below.

```
/** @param start starting point for values to be checked
 *      Precondition: start > 0
 *      @param num the size of the array to be returned
 *      Precondition: num > 0
 *      @return an array containing the first num integers ≥ start that are self-divisors
 */
public static int[] firstNumSelfDivisors(int start, int num)
```

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. Consider a system for processing student test scores. The following class will be used as part of this system and contains a student's name and the student's answers for a multiple-choice test. The answers are represented as strings of length one with an omitted answer being represented by a string containing a single question mark ("?"). These answers are stored in an `ArrayList` in which the position of the answer corresponds to the question number on the test (question numbers start at 0). A student's score on the test is computed by comparing the student's answers with the corresponding answers in the answer key for the test. One point is awarded for each correct answer and  $\frac{1}{4}$  of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

```
public class StudentAnswerSheet
{
    private ArrayList<String> answers; // the list of the student's answers

    /**
     * @param key the list of correct answers, represented as strings of length one
     *           Precondition: key.size() is equal to the number of answers in this answer sheet
     *           @return this student's test score
     */
    public double getScore(ArrayList<String> key)
    { /* to be implemented in part (a) */ }

    /**
     * @return the name of the student
     */
    public String getName()
    { /* implementation not shown */ }

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

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The following table shows an example of an answer key, a student's answers, and the corresponding point values that would be awarded for the student's answers. In this example, there are six correct answers, three incorrect answers, and one omitted answer. The student's score is  $((6 * 1) - (3 * 0.25)) = 5.25$ .

Question number	0	1	2	3	4	5	6	7	8	9
key	"A"	"C"	"D"	"E"	"B"	"C"	"E"	"B"	"B"	"C"
answers	"A"	"B"	"D"	"E"	"A"	"C"	"?"	"B"	"D"	"C"
Points awarded	1	-0.25	1	1	-0.25	1	0	1	-0.25	1

- (a) Write the `StudentAnswerSheet` method `getScore`. The parameter passed to method `getScore` is an `ArrayList` of strings representing the correct answer key for the test being scored. The method computes and returns a `double` that represents the score for the student's test answers when compared with the answer key. One point is awarded for each correct answer and  $\frac{1}{4}$  of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

Complete method `getScore` below.

```

/** @param key the list of correct answers, represented as strings of length one
 *      Precondition: key.size() is equal to the number of answers in this answer sheet
 *      @return this student's test score
 */
public double getScore(ArrayList<String> key)

```

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Consider the following class that represents the test results of a group of students that took a multiple-choice test.

```
public class TestResults
{
    private ArrayList<StudentAnswerSheet> sheets;

    /** Precondition: sheets.size() > 0;
     *               all answer sheets in sheets have the same number of answers
     * @param key the list of correct answers represented as strings of length one
     * Precondition: key.size() is equal to the number of answers
     *               in each of the answer sheets in sheets
     * @return the name of the student with the highest score
     */
    public String highestScoringStudent(ArrayList<String> key)
    { /* to be implemented in part (b) */ }

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

Write the `TestResults` method `highestScoringStudent`, which returns the name of the student who received the highest score on the test represented by the parameter `key`. If there is more than one student with the highest score, the name of any one of these highest-scoring students may be returned. You may assume that the size of each answer sheet represented in the `ArrayList` `sheets` is equal to the size of the `ArrayList` `key`.

In writing `highestScoringStudent`, assume that `getScore` works as specified, regardless of what you wrote in part (a).

Complete method `highestScoringStudent` below.

```
/** Precondition: sheets.size() > 0;
 *               all answer sheets in sheets have the same number of answers
 * @param key the list of correct answers represented as strings of length one
 * Precondition: key.size() is equal to the number of answers
 *               in each of the answer sheets in sheets
 * @return the name of the student with the highest score
 */
public String highestScoringStudent(ArrayList<String> key)
```

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.

The `GameState` interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.

The `GameState` interface specifies these methods. The `Player` class will be described in part (a).

```
public interface GameState
{
    /** @return true if the game is in an ending state;
     *         false otherwise
     */
    boolean isGameOver();

    /**
     * Precondition: isGameOver() returns true
     * @return the player that won the game or null if there was no winner
     */
    Player getWinner();

    /**
     * Precondition: isGameOver() returns false
     * @return the player who is to make the next move
     */
    Player getCurrentPlayer();

    /**
     * @return a list of valid moves for the current player;
     *         the size of the returned list is 0 if there are no valid moves.
     */
    ArrayList<String> getCurrentMoves();

    /**
     * Updates game state to reflect the effect of the specified move.
     * @param move a description of the move to be made
     */
    void makeMove(String move);

    /**
     * @return a string representing the current GameState
     */
    String toString();
}
```

The `makeMove` method makes the move specified, updating the state of the game being played. Its parameter is a `String` that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like "X-1-1", indicating an X is put in the position (1, 1).

## 2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) The `Player` class provides a method for selecting the next move. By extending this class, different playing strategies can be modeled.

```
public class Player
{
    private String name;      // name of this player

    public Player(String aName)
    {   name = aName;   }

    public String getName()
    {   return name;   }

    /**
     * This implementation chooses the first valid move.
     * Override this method in subclasses to define players with other strategies.
     * @param state the current state of the game; its current player is this player.
     * @return a string representing the move chosen;
     *         "no move" if no valid moves for the current player.
     */
    public String getNextMove(GameState state)
    {   /* implementation not shown */   }
}
```

The method `getNextMove` returns the next move to be made as a string, using the same format as that used by `makeMove` in `GameState`. Depending on how the `getNextMove` method is implemented, a player can exhibit different game-playing strategies.

Write the complete class declaration for a `RandomPlayer` class that is a subclass of `Player`. The class should have a constructor whose `String` parameter is the player's name. It should override the `getNextMove` method to randomly select one of the valid moves in the given game state. If there are no valid moves available for the player, the string "no move" should be returned.

## **2007 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) The GameDriver class is used to manage the state of the game during game play. The GameDriver class can be written without knowing details about the game being played.

```
public class GameDriver
{
    private GameState state; // the current state of the game

    public GameDriver(GameState initial)
    {   state = initial;   }

    /** Plays an entire game, as described in the problem description
     */
    public void play()
    {   /* to be implemented in part (b) */ }

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

Write the GameDriver method play. This method should first print the initial state of the game. It should then repeatedly determine the current player and that player's next move, print both the player's name and the chosen move, and make the move. When the game is over, it should stop making moves and print either the name of the winner and the word "wins" or the message "Game ends in a draw" if there is no winner. You may assume that the GameState makeMove method has been implemented so that it will properly handle any move description returned by the Player getNextMove method, including the string "no move".

Complete method play below.

```
/** Plays an entire game, as described in the problem description
 */
public void play()
```

**STOP**

**END OF EXAM**



## **AP® Computer Science A 2007 Scoring Guidelines**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2007 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.

Permission to use copyrighted College Board materials may be requested online at:  
[www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).  
AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**AP® COMPUTER SCIENCE A  
2007 SCORING GUIDELINES**

**Question 1: Self Divisor**

<b>Part A:</b>	isSelfDivisor	<b>4 points</b>
----------------	---------------	-----------------

- +2 loop over digits
    - +1 access digit *in context of loop*
      - +1/2 attempt (`number % ?` or successfully convert to string represent)
      - +1/2 correct
    - +1 process all digits
      - +1/2 attempt (process multiple digits)
      - +1/2 correct
  - +2 classify number
    - +1/2 return false if find 0 digit
    - +1/2 test if divisible (`number % digit`)
    - +1/2 return false if find non-divisor digit
    - +1/2 return true self divisor
- ] *lose both of these if return a value in both cases of an if-else*

<b>Part B:</b>	firstNumSelfDivisors	<b>5 points</b>
----------------	----------------------	-----------------

- +1 initialize
  - +1/2 create and initialize array of size num
  - +1/2 create and initialize index counter
- +3 1/2 loop to find self divisors
  - +1/2 iterate through numbers beginning with start
  - +1/2 call `isSelfDivisor` on number
  - +1 1/2 add self divisor to array
    - +1/2 attempt (store self divisor in some array index)
    - +1 correct (store in correct index, including increment)
  - +1 loop and store num values in array
    - +1/2 attempt (must reference index counter and num)
    - +1/2 correct
- +1/2 return array (lose this if return first time through loop)

**AP® COMPUTER SCIENCE A  
2007 SCORING GUIDELINES**

**Question 3: Answer Sheets**

<b>Part A:</b>	<b>getScore</b>	<b>4 1/2 points</b>
----------------	-----------------	---------------------

- +1/2 initialize score (a double) or right/wrong counters
- +1 1/2 loop over either answers or key
  - +1/2 reference answers or key in loop body
  - +1/2 correctly access answers or key element in loop body
  - +1/2 access all answers or key elements
- +2 calculate score
  - +1/2 attempt to compare an answers element and a key element (== ok)
  - +1/2 correctly compare corresponding elements using equals
  - +1/2 add 1 to score if and only if equal
  - +1/2 subtract 1/4 from score if and only if not equal and answer not "?"
- +1/2 return calculated score

<b>Part B:</b>	<b>highestScoringStudent</b>	<b>4 1/2 points</b>
----------------	------------------------------	---------------------

- +1 1/2 loop over sheets
  - +1/2 reference sheets in loop body
  - +1/2 correctly access sheets element in context of loop
  - +1/2 access all elements of sheets
- +2 determine highest score
  - +1/2 get student score (call getScore (key) on a sheets element)
  - +1/2 compare student score with highest so far (in context of loop)
  - +1 correctly identify highest score (lose this if use constant for initial high)
- +1 return name
  - +1/2 access name (call getName on highest)
  - +1/2 return name

**AP® COMPUTER SCIENCE A  
2007 SCORING GUIDELINES**

**Question 4: Game Design (Design)**

<b>Part A:</b>	RandomPlayer	<b>4 points</b>
----------------	--------------	-----------------

+1/2 class RandomPlayer extends Player

+1 constructor

+1/2 public RandomPlayer(String aName)

+1/2 super(aName)

+2 1/2 getNextMove

+1/2 state.getCurrentMoves()

+1 if no moves

+1/2 test if size = 0

+1/2 return "no move" only if 0 moves

+1 if moves

+1/2 select random move index

+1/2 return random move

<b>Part B:</b>	play	<b>5 points</b>
----------------	------	-----------------

+1/2 print initial state (OK to print in loop)

+3 make repeated moves

+1 repeat until state.isGameOver()

+1/2 state.getCurrentPlayer()

+1/2 player.getNextMove(state)

+1/2 display player and move

+1/2 make move

+1 1/2 determine winner

+1/2 state.getWinner()

+1/2 display message if draw (if getWinner returns null)

+1/2 display message if winner

] lose both if done before game ends



## **AP® Computer Science A 2006 Free-Response Questions**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2006 The College Board. All rights reserved. College Board, AP Central, APCD, Advanced Placement Program, AP, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service, CollegeEd, connect to college success, MyRoad, SAT Professional Development, SAT Readiness Program, and Setting the Cornerstones are trademarks owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**COMPUTER SCIENCE A**

**SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

**Notes:**

- Assume that the classes listed in the Quick Reference found in the Appendix 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.

## 2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

1. An appointment scheduling system is represented by the following three classes: `TimeInterval`, `Appointment`, and `DailySchedule`. In this question, you will implement one method in the `Appointment` class and two methods in the `DailySchedule` class.

A `TimeInterval` object represents a period of time. The `TimeInterval` class provides a method to determine if another time interval overlaps with the time interval represented by the current `TimeInterval` object. An `Appointment` object contains a time interval for the appointment and a method that determines if there is a time conflict between the current appointment and another appointment. The declarations of the `TimeInterval` and `Appointment` classes are shown below.

```
public class TimeInterval
{
    // returns true if interval overlaps with this TimeInterval;
    // otherwise, returns false
    public boolean overlapsWith(TimeInterval interval)
    { /* implementation not shown */ }

    // There may be fields, constructors, and methods that are not shown.
}

public class Appointment
{
    // returns the time interval of this Appointment
    public TimeInterval getTime()
    { /* implementation not shown */ }

    // returns true if the time interval of this Appointment
    // overlaps with the time interval of other;
    // otherwise, returns false
    public boolean conflictsWith(Appointment other)
    { /* to be implemented in part (a) */ }

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

- (a) Write the `Appointment` method `conflictsWith`. If the time interval of the current appointment overlaps with the time interval of the appointment `other`, method `conflictsWith` should return `true`, otherwise, it should return `false`.

Complete method `conflictsWith` below.

```
// returns true if the time interval of this Appointment
// overlaps with the time interval of other;
// otherwise, returns false
public boolean conflictsWith(Appointment other)
```

## 2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) A DailySchedule object contains a list of nonoverlapping Appointment objects. The DailySchedule class contains methods to clear all appointments that conflict with a given appointment and to add an appointment to the schedule.

```
public class DailySchedule
{
    // contains Appointment objects, no two Appointments overlap
    private ArrayList apptList;

    public DailySchedule()
    {   apptList = new ArrayList();   }

    // removes all appointments that overlap the given Appointment
    // postcondition: all appointments that have a time conflict with
    //                   appt have been removed from this DailySchedule
    public void clearConflicts(Appointment appt)
    {   /* to be implemented in part (b) */   }

    // if emergency is true, clears any overlapping appointments and adds
    // appt to this DailySchedule; otherwise, if there are no conflicting
    // appointments, adds appt to this DailySchedule;
    // returns true if the appointment was added;
    // otherwise, returns false
    public boolean addAppt(Appointment appt, boolean emergency)
    {   /* to be implemented in part (c) */   }

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

Write the DailySchedule method clearConflicts. Method clearConflicts removes all appointments that conflict with the given appointment.

In writing method clearConflicts, you may assume that conflictsWith works as specified, regardless of what you wrote in part (a).

Complete method clearConflicts below.

```
// removes all appointments that overlap the given Appointment
// postcondition: all appointments that have a time conflict with
//                   appt have been removed from this DailySchedule
public void clearConflicts(Appointment appt)
```

## **2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (c) Write the `DailySchedule` method `addAppt`. The parameters to method `addAppt` are an appointment and a boolean value that indicates whether the appointment to be added is an emergency. If the appointment is an emergency, the schedule is cleared of all appointments that have a time conflict with the given appointment and the appointment is added to the schedule. If the appointment is not an emergency, the schedule is checked for any conflicting appointments. If there are no conflicting appointments, the given appointment is added to the schedule. Method `addAppt` returns `true` if the appointment was added to the schedule; otherwise, it returns `false`.

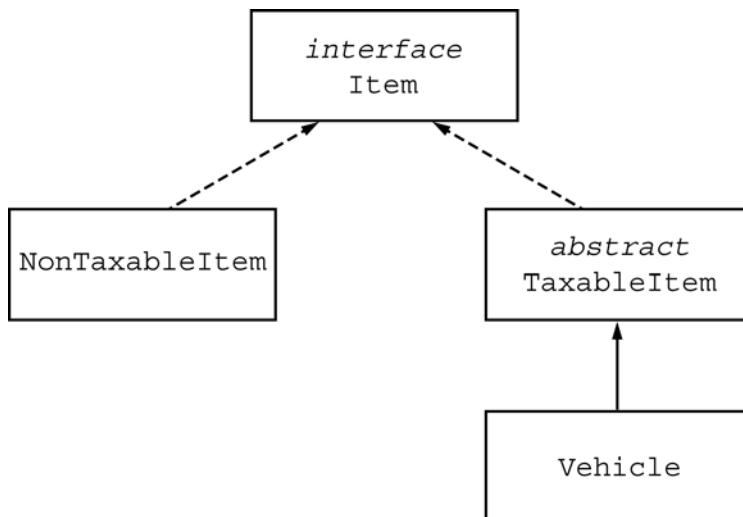
In writing method `addAppt`, you may assume that `conflictsWith` and `clearConflicts` work as specified, regardless of what you wrote in parts (a) and (b).

Complete method `addAppt` below.

```
// if emergency is true, clears any overlapping appointments and adds  
// appt to this DailySchedule; otherwise, if there are no conflicting  
// appointments, adds appt to this DailySchedule;  
// returns true if the appointment was added;  
// otherwise, returns false  
public boolean addAppt(Appointment appt, boolean emergency)
```

## 2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. A set of classes is used to represent various items that are available for purchase. Items are either taxable or nontaxable. The purchase price of a taxable item is computed from its list price and its tax rate. The purchase price of a nontaxable item is simply its list price. Part of the class hierarchy is shown in the diagram below.



The definitions of the `Item` interface and the `TaxableItem` class are shown below.

```
public interface Item
{
    double purchasePrice();
}

public abstract class TaxableItem implements Item
{
    private double taxRate;

    public abstract double getListPrice();

    public TaxableItem(double rate)
    {   taxRate = rate;   }

    // returns the price of the item including the tax
    public double purchasePrice()
    {   /* to be implemented in part (a) */   }
}
```

## **2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the `TaxableItem` method `purchasePrice`. The purchase price of a `TaxableItem` is its list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (representing 10%), the purchase price of an item with a list price of \$6.50 would be \$7.15.

Complete method `purchasePrice` below.

```
// returns the price of the item including the tax
public double purchasePrice()
```

- (b) Create the `Vehicle` class, which extends the `TaxableItem` class. A vehicle has two parts to its list price: a dealer cost and dealer markup. The list price of a vehicle is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of \$20,000.00, a dealer markup of \$2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be \$22,500.00 and the purchase price (including tax) would be \$24,750.00. If the dealer markup were changed to \$1,000.00, then the list price of the vehicle would be \$21,000.00 and the purchase price would be \$23,100.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters. Provide any private instance variables needed and implement all necessary methods. Also provide a public method `changeMarkup`, which changes the dealer markup to the value of its parameter.

## 2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

3. Consider the following incomplete class that stores information about a customer, which includes a name and unique ID (a positive integer). To facilitate sorting, customers are ordered alphabetically by name. If two or more customers have the same name, they are further ordered by ID number. A particular customer is "greater than" another customer if that particular customer appears later in the ordering than the other customer.

```
public class Customer
{
    // constructs a Customer with given name and ID number
    public Customer(String name, int idNum)
    { /* implementation not shown */ }

    // returns the customer's name
    public String getName()
    { /* implementation not shown */ }

    // returns the customer's id
    public int getID()
    { /* implementation not shown */ }

    // returns 0 when this customer is equal to other;
    // a positive integer when this customer is greater than other;
    // a negative integer when this customer is less than other
    public int compareCustomer(Customer other)
    { /* to be implemented in part (a) */ }

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

- (a) Write the `Customer` method `compareCustomer`, which compares this customer to a given customer, `other`. Customers are ordered alphabetically by name, using the `compareTo` method of the `String` class. If the names of the two customers are the same, then the customers are ordered by ID number. Method `compareCustomer` should return a positive integer if this customer is greater than `other`, a negative integer if this customer is less than `other`, and 0 if they are the same.

For example, suppose we have the following `Customer` objects.

```
Customer c1 = new Customer("Smith", 1001);
Customer c2 = new Customer("Anderson", 1002);
Customer c3 = new Customer("Smith", 1003);
```

The following table shows the result of several calls to `compareCustomer`.

<u>Method Call</u>	<u>Result</u>
<code>c1.compareCustomer(c1)</code>	0
<code>c1.compareCustomer(c2)</code>	a positive integer
<code>c1.compareCustomer(c3)</code>	a negative integer

## **2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method compareCustomer below.

```
// returns 0 when this customer is equal to other;
//   a positive integer when this customer is greater than other;
//   a negative integer when this customer is less than other
public int compareCustomer(Customer other)
```

## **2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (b) A company maintains customer lists where each list is a sorted array of customers stored in ascending order by customer. A customer may appear in more than one list, but will not appear more than once in the same list.

Write method `prefixMerge`, which takes three array parameters. The first two arrays, `list1` and `list2`, represent existing customer lists. It is possible that some customers are in both arrays. The third array, `result`, has been instantiated to a length that is no longer than either of the other two arrays and initially contains `null` values. Method `prefixMerge` uses an algorithm similar to the merge step of a Mergesort to fill the array `result`. Customers are copied into `result` from the beginning of `list1` and `list2`, merging them in ascending order until all positions of `result` have been filled. Customers who appear in both `list1` and `list2` will appear at most once in `result`.

For example, assume that three arrays have been initialized as shown below.

list1	Arthur 4920	Burton 3911	Burton 4944	Franz 1692	Horton 9221	Jones 5554	Miller 9360	Nguyen 4339
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list2	Aaron 1729	Baker 2921	Burton 3911	Dillard 6552	Jones 5554	Miller 9360	Noble 3335	
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	
result	null	null	null	null	null	null		
	[0]	[1]	[2]	[3]	[4]	[5]		

In this example, the array `result` must contain the following values after the call `prefixMerge(list1, list2, result)`.

result	Aaron 1729	Arthur 4920	Baker 2921	Burton 3911	Burton 4944	Dillard 6552
	[0]	[1]	[2]	[3]	[4]	[5]

In writing `prefixMerge`, you may assume that `compareCustomer` works as specified, regardless of what you wrote in part (a). Solutions that create any additional data structures holding multiple objects (e.g., arrays, `ArrayLists`, etc.) will not receive full credit.

## **2006 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Complete method `prefixMerge` below.

```
// fills result with customers merged from the
// beginning of list1 and list2;
// result contains no duplicates and is sorted in
// ascending order by customer
// precondition: result.length > 0;
//                      list1.length >= result.length;
//                      list1 contains no duplicates;
//                      list2.length >= result.length;
//                      list2 contains no duplicates;
//                      list1 and list2 are sorted in
//                      ascending order by customer
// postcondition: list1, list2 are not modified
public static void prefixMerge(Customer[] list1,
                               Customer[] list2,
                               Customer[] result)
```



## **AP® Computer Science A 2006 Scoring Guidelines**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2006 The College Board. All rights reserved. College Board, AP Central, APCD, Advanced Placement Program, AP, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Board. Admitted Class Evaluation Service, CollegeEd, connect to college success, MyRoad, SAT Professional Development, SAT Readiness Program, and Setting the Cornerstones are trademarks owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: [www.collegeboard.com/inquiry/cbpermit.html](http://www.collegeboard.com/inquiry/cbpermit.html).

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).

AP Central is the official online home for the AP Program: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**AP® COMPUTER SCIENCE A  
2006 SCORING GUIDELINES**

**Question 1: Daily Schedule**

<b>Part A:</b>	<code>conflictsWith</code>	<b>1 1/2 points</b>
----------------	----------------------------	---------------------

- +1/2 call `OBJ1.overlapsWith(OBJ2)`
- +1/2 access `getTime` of other and `this`
- +1/2 return correct value

<b>Part B:</b>	<code>clearConflicts</code>	<b>3 points</b>
----------------	-----------------------------	-----------------

- +2 loop over `apptList`
  - +1/2 reference `apptList` in loop body
  - +1/2 access appointment *in context of loop* (`apptList.get(i)`)
  - +1 access all appointments (cannot skip entries after a removal)
- +1 remove conflicts *in context of loop*
  - +1/2 determine when conflict exists (must call `conflictsWith`)
  - +1/2 remove all conflicting appointments (and no others)

<b>Part C:</b>	<code>addAppt</code>	<b>4 1/2 points</b>
----------------	----------------------	---------------------

- +1/2 test if emergency (*may limit to when emergency AND conflict exists*)
- +1/2 clear conflicts if and only if emergency
  - (must not reimplement `clearConflicts` code)
- +1/2 add `appt` if emergency
- +2 non-emergency case
  - +1/2 loop over `apptList` (must reference `apptList` in body)
  - +1/2 access `apptList` element and check for `appt` conflicts *in context of loop*
  - +1/2 exit loop with state (conflict / no conflict) correctly determined
    - (*includes loop bound*)
  - +1/2 add `appt` if and only if no conflict
- +1 return true if any appointment added, false otherwise (must return both)

**Usage:** -1 if loop structure results in failure to handle empty `apptList`

**AP<sup>®</sup> COMPUTER SCIENCE A  
2006 SCORING GUIDELINES**

**Question 2: Taxable Items (Design)**

<b>Part A:</b>	<code>purchasePrice</code>	<b>2 1/2 points</b>
----------------	----------------------------	---------------------

- +1 call `getListPrice()`
- +1 calculate correct purchase price (*no penalty if truncate/round to 2 decimal places*)
- +1/2 return calculated price

<b>Part B:</b>	<code>Vehicle</code>	<b>6 1/2 points</b>
----------------	----------------------	---------------------

- +1/2 class `Vehicle` extends `TaxableItem`
- +1/2 private double `dealerCost`
- +1/2 private double `dealerMarkup` (*no penalty if also store tax in field*)
- +2 1/2 constructor
  - +1/2 `Vehicle(double ?, double ?, double ?)`  
int/float (*OK if match fields*)
  - +1 call parent constructor
    - +1/2 attempt using `super`
    - +1/2 correct call: `super(rate)` (*note: must be first line in method*)
  - +1 initialize dealer cost and markup fields
    - +1/2 attempt (must use parameters on RHS or in mutator call)
    - +1/2 correct
- +1 `changeMarkup`
  - +1/2 public void `changeMarkup(double ?)`  
int/float (*OK if matches field; no penalty if returns reasonable value*)
  - +1/2 assign parameter to markup field
- +1 1/2 `getListPrice`
  - +1 public double `getListPrice()`
  - +1/2 return sum of dealer cost and markup fields

**Note:** -1 usage if reimplement `purchasePrice` to do anything other than  
`return super.purchasePrice();`

**AP<sup>®</sup> COMPUTER SCIENCE A  
2006 SCORING GUIDELINES**

**Question 3: Customer List**

<b>Part A:</b>	<code>compareCustomer</code>	<b>3 points</b>
----------------	------------------------------	-----------------

- +1 1/2 perform comparison
  - +1/2 attempt (must call `OBJ1.compareTo(OBJ2)`)
  - +1/2 correctly access and compare names
  - +1/2 correctly access and compare IDs
- +1/2 return 0 if and only if `this = other`
- +1/2 return positive if and only if `this > other`
- +1/2 return negative if and only if `this < other`

<b>Part B:</b>	<code>prefixMerge</code>	<b>6 points</b>
----------------	--------------------------	-----------------

- +1/2 initialize unique variables to index fronts of arrays
- +1 1/2 loop over arrays to fill result
  - +1/2 attempt (must reference `list1` and `list2` inside loop)
  - +1 correct (lose this if add too few or too many `Customer` elements)
- +1 1/2 compare array fronts (in context of loop)
  - +1/2 attempt (must call `compareCustomer` on array elements)
  - +1 correctly compare front `Customer` elements
- +1 1/2 duplicate entries
  - +1/2 check if duplicate entries found
  - +1/2 if duplicates, copy only one to `result` (without use of additional structure)
  - +1/2 update indices into both arrays (`list1` and `list2`)
- +1 nonduplicate entries
  - +1/2 copy only smallest entry to `result` (without use of additional structure)
  - +1/2 update index into that array only (`list1` or `list2`)

Note: Solution may use constants as returned from part A.

**Usage:** -1/2 `compareTo` instead of `compareCustomer` for `Customer` objects



## **AP® Computer Science A 2005 Free-Response Questions**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2005 by College Board. All rights reserved. College Board, AP Central, APCD, Advanced Placement Program, AP, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Entrance Examination Board. Admitted Class Evaluation Service, CollegeEd, Connect to college success, MyRoad, SAT Professional Development, SAT Readiness Program, and Setting the Cornerstones are trademarks owned by the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: <http://www.collegeboard.com/inquiry/cbpermit.html>.

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).

AP Central is the official online home for the AP Program and Pre-AP: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

**COMPUTER SCIENCE A**

**SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

**Notes:**

- Assume that the classes listed in the Quick Reference found in the Appendix 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.

## 2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

1. In this question, you will implement two methods for a class `Hotel` that is part of a hotel reservation system. The `Hotel` class uses the `Reservation` class shown below. A `Reservation` is for the person and room number specified when the `Reservation` is constructed.

```
public class Reservation
{
    public Reservation(String guestName, int roomNumber)
    { /* implementation not shown */ }

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

    // private data and other methods not shown
}
```

An incomplete declaration for the `Hotel` class is shown below. Each hotel in the hotel reservation system has rooms numbered 0, 1, 2, . . . , up to the last room number in the hotel. For example, a hotel with 100 rooms would have rooms numbered 0, 1, 2, . . . , 99.

```
public class Hotel
{
    private Reservation[] rooms;
    // each element corresponds to a room in the hotel;
    // if rooms[index] is null, the room is empty;
    // otherwise, it contains a reference to the Reservation
    // for that room, such that
    // rooms[index].getRoomNumber() returns index

    private ArrayList waitList;
    // contains names of guests who have not yet been
    // assigned a room because all rooms are full

    // if there are any empty rooms (rooms with no reservation),
    // then create a reservation for an empty room for the
    // specified guest and return the new Reservation;
    // otherwise, add the guest to the end of waitList
    // and return null
    public Reservation requestRoom(String guestName)
    { /* to be implemented in part (a) */ }

    // release the room associated with parameter res, effectively
    // canceling the reservation;
    // if any names are stored in waitList, remove the first name
    // and create a Reservation for this person in the room
    // reserved by res; return that new Reservation;
    // if waitList is empty, mark the room specified by res as empty and
    // return null
    // precondition: res is a valid Reservation for some room
    //               in this hotel
    public Reservation cancelAndReassign(Reservation res)
    { /* to be implemented in part (b) */ }

    // constructors and other methods not shown
}
```

Copyright © 2005 by College Entrance Examination Board. All rights reserved.  
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for AP students and parents).

**GO ON TO THE NEXT PAGE.**

## **2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the Hotel method `requestRoom`. Method `requestRoom` attempts to reserve a room in the hotel for a given guest. If there are any empty rooms in the hotel, one of them will be assigned to the named guest and the newly created reservation is returned. If there are no empty rooms, the guest is added to the end of the waiting list and `null` is returned.

Complete method `requestRoom` below.

```
// if there are any empty rooms (rooms with no reservation),  
// then create a reservation for an empty room for the  
// specified guest and return the new Reservation;  
// otherwise, add the guest to the end of waitList  
// and return null  
public Reservation requestRoom(String guestName)
```

- (b) Write the Hotel method `cancelAndReassign`. Method `cancelAndReassign` releases a previous reservation. If the waiting list for the hotel contains any names, the vacated room is reassigned to the first person at the beginning of the list. That person is then removed from the waiting list and the newly created reservation is returned. If no one is waiting, the room is marked as empty and `null` is returned.

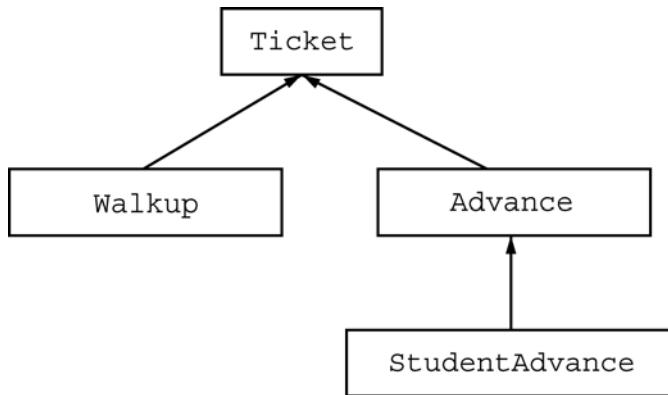
In writing `cancelAndReassign` you may call any accessible methods in the `Reservation` and `Hotel` classes. Assume that these methods work as specified.

Complete method `cancelAndReassign` below.

```
// release the room associated with parameter res, effectively  
// canceling the reservation;  
// if any names are stored in waitList, remove the first name  
// and create a Reservation for this person in the room  
// reserved by res; return that new Reservation;  
// if waitList is empty, mark the room specified by res as empty and  
// return null  
// precondition: res is a valid Reservation for some room  
//                 in this hotel  
public Reservation cancelAndReassign(Reservation res)
```

## 2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. A set of classes is used to handle the different ticket types for a theater. The class hierarchy is shown in the following diagram.



All tickets have a serial number and a price. The class `Ticket` is specified as an `abstract` class as shown in the following declaration.

```
public abstract class Ticket
{
    private int serialNumber;      // unique ticket id number

    public Ticket()
    {   serialNumber = getNextSerialNumber();   }

    // returns the price for this ticket
    public abstract double getPrice();

    // returns a string with information about the ticket
    public String toString()
    {
        return "Number: " + serialNumber + "\nPrice: " + getPrice();
    }

    // returns a new unique serial number
    private static int getNextSerialNumber()
    { /* implementation not shown */ }
}
```

## **2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

Each ticket has a unique serial number that is assigned when the ticket is constructed. For all ticket classes, the `toString` method returns a string containing the information for that ticket. Three additional classes are used to represent the different types of tickets and are described in the table below.

Class	Description	Sample <code>toString</code> Output
Walkup	These tickets are purchased on the day of the event and cost 50 dollars.	Number: 712 Price: 50
Advance	Tickets purchased ten or more days in advance cost 30 dollars. Tickets purchased fewer than ten days in advance cost 40 dollars.	Number: 357 Price: 40
StudentAdvance	These tickets are a type of Advance ticket that costs half of what that Advance ticket would normally cost.	Number: 134 Price: 15 (student ID required)

Using the class hierarchy and specifications given above, you will write complete class declarations for the `Advance` and `StudentAdvance` classes.

- (a) Write the complete class declaration for the class `Advance`. Include all necessary instance variables and implementations of its constructor and method(s). The constructor should take a parameter that indicates the number of days in advance that this ticket is being purchased. Tickets purchased ten or more days in advance cost \$30; tickets purchased nine or fewer days in advance cost \$40.
- (b) Write the complete class declaration for the class `StudentAdvance`. Include all necessary instance variables and implementations of its constructor and method(s). The constructor should take a parameter that indicates the number of days in advance that this ticket is being purchased. The `toString` method should include a notation that a student ID is required for this ticket. A `StudentAdvance` ticket costs half of what that `Advance` ticket would normally cost. If the pricing scheme for `Advance` tickets changes, the `StudentAdvance` price should continue to be computed correctly with no code modifications to the `StudentAdvance` class.

## 2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. Consider a grade-averaging scheme in which the final average of a student's scores is computed differently from the traditional average if the scores have "improved." Scores have improved if each score is greater than or equal to the previous score. The final average of the scores is computed as follows.

A student has  $n$  scores indexed from 0 to  $n-1$ . If the scores have improved, only those scores with indexes greater than or equal to  $n/2$  are averaged. If the scores have not improved, all the scores are averaged.

The following table shows several lists of scores and how they would be averaged using the scheme described above.

<u>Student Scores</u>	<u>Improved?</u>	<u>Final Average</u>
50, 50, 20, 80, 53	No	$(50 + 50 + 20 + 80 + 53) / 5.0 = 50.6$
20, 50, 50, 53, 80	Yes	$(50 + 53 + 80) / 3.0 = 61.0$
20, 50, 50, 80	Yes	$(50 + 80) / 2.0 = 65.0$

Consider the following incomplete `StudentRecord` class declaration. Each `StudentRecord` object stores a list of that student's scores and contains methods to compute that student's final average.

```
public class StudentRecord
{
    private int[] scores; // contains scores.length values
                          // scores.length > 1

    // constructors and other data fields not shown

    // returns the average (arithmetic mean) of the values in scores
    // whose subscripts are between first and last, inclusive
    // precondition: 0 <= first <= last < scores.length
    private double average(int first, int last)
    { /* to be implemented in part (a) */ }

    // returns true if each successive value in scores is greater
    // than or equal to the previous value;
    // otherwise, returns false
    private boolean hasImproved()
    { /* to be implemented in part (b) */ }

    // if the values in scores have improved, returns the average
    // of the elements in scores with indexes greater than or equal
    // to scores.length/2;
    // otherwise, returns the average of all of the values in scores
    public double finalAverage()
    { /* to be implemented in part (c) */ }
}
```

## **2005 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

- (a) Write the `StudentRecord` method `average`. This method returns the average of the values in `scores` given a starting and an ending index.

Complete method `average` below.

```
// returns the average (arithmetic mean) of the values in scores  
// whose subscripts are between first and last, inclusive  
// precondition: 0 <= first <= last < scores.length  
private double average(int first, int last)
```

- (b) Write the `StudentRecord` method `hasImproved`.

Complete method `hasImproved` below.

```
// returns true if each successive value in scores is greater  
// than or equal to the previous value;  
// otherwise, returns false  
private boolean hasImproved()
```

- (c) Write the `StudentRecord` method `finalAverage`.

In writing `finalAverage`, you must call the methods defined in parts (a) and (b). Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b).

Complete method `finalAverage` below.

```
// if the values in scores have improved, returns the average  
// of the elements in scores with indexes greater than or equal  
// to scores.length/2;  
// otherwise, returns the average of all of the values in scores  
public double finalAverage()
```

**END OF EXAM**



## **AP® Computer Science A 2005 Scoring Guidelines**

### **The College Board: Connecting Students to College Success**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2005 by College Board. All rights reserved. College Board, AP Central, APCD, Advanced Placement Program, AP, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Entrance Examination Board. Admitted Class Evaluation Service, CollegeEd, Connect to college success, MyRoad, SAT Professional Development, SAT Readiness Program, and Setting the Cornerstones are trademarks owned by the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Other products and services may be trademarks of their respective owners. Permission to use copyrighted College Board materials may be requested online at: <http://www.collegeboard.com/inquiry/cbpermit.html>.

Visit the College Board on the Web: [www.collegeboard.com](http://www.collegeboard.com).

AP Central is the official online home for the AP Program and Pre-AP: [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**AP® COMPUTER SCIENCE A  
2005 SCORING GUIDELINES**

**2005 A Question 1: Hotel Reservation**

<b>Part A:</b>	<code>requestRoom</code>	<b>4 points</b>
----------------	--------------------------	-----------------

- +1 loop over `rooms`
  - +1/2 attempt (must reference multiple elements of `rooms` in body)
  - +1/2 correct
- +1/2 test correct array entry for null (in context of loop)
- +1 1/2 handle new reservation (in context of a loop)
  - +1/2 attempt to create new reservation (some sense of `Reservation` construction)
  - +1/2 correctly create reservation (if add to `rooms`, must be in null location & assignment correct)
  - +1/2 return reservation (only if null entry)
- +1 handle wait list after loop or at appropriate time (only if full)
  - +1/2 add new guest to end of `waitlist` only once
  - +1/2 return null

<b>Part B:</b>	<code>cancelAndReassign</code>	<b>5 points</b>
----------------	--------------------------------	-----------------

- +1 look up room number
  - +1/2 attempt (must call `res.getRoomNumber()` or use loop to find `res`)
  - +1/2 correct (must call `res.getRoomNumber()`)
- +1/2 test `waitlist` to see if empty
- +2 1/2 handle nonempty `waitList`
  - +1/2 get *first* entry from `waitList` (only if `waitlist` is not empty)
  - +1/2 create new `Reservation`      }      *can get these points by*
  - +1/2 assign `Reservation` to correct room      }      *correctly calling requestRoom*
  - +1/2 remove only first entry from `waitlist` (only if `waitlist` is not empty)
  - +1/2 return new `Reservation` (only if `waitlist` is not empty)
- +1 handle empty case
  - +1/2 assign null to room (only if `waitList` is empty)
  - +1/2 return null (only if `waitList` is empty)

Note: If access using `get` on `rooms` is done more than once, deduct 1/2 usage point, not correctness (ditto for `set` on `rooms`).

**AP® COMPUTER SCIENCE A  
2005 SCORING GUIDELINES**

**2005 A Question 2: Ticket Sales**

<b>Part A:</b>	Advance	<b>3 1/2 points</b>
----------------	---------	---------------------

- +1/2 class Advance extends Ticket (no abstract)
- +1/2 private data field (either days or price)
- +1 1/2 constructor
  - +1/2 correct header
  - +1 correctly assign data field(s) (lose if reference to super's private data)
- +1 getPrice
  - +1/2 correct header (must be public & double, no abstract, no parameters)
  - +1/2 return correct price

<b>Part B:</b>	StudentAdvance	<b>5 1/2 points</b>
----------------	----------------	---------------------

- +1/2 class StudentAdvance extends Advance
- +1 1/2 constructor
  - +1/2 correct header
  - +1/2 attempt to call super
  - +1/2 correct call to super
- +2 getPrice
  - +1/2 correct header (must be public & double, no abstract, no parameters)
  - +1 call super.getPrice()
  - +1/2 calculate and return correct price
- +1 1/2 toString
  - +1/2 call super.toString()
  - +1 return string with correct phrase concatenated (lose this with a reference to super class's private data)

Usage: -1/2 in part A if super() appears in the constructor and it is not the first statement executed.

**AP® COMPUTER SCIENCE A  
2005 SCORING GUIDELINES**

**2005 A Question 4: Improving Grades**

<b>Part A:</b>	<b>average</b>	<b>3 points</b>
----------------	----------------	-----------------

- +1/2 initialize sum
- +1 loop over scores
  - +1/2 attempt (must reference `scores` in body)
  - +1/2 correct (from first to last)
- +1/2 add score to sum (in context of loop)
- +1 calculate and return average
  - +1/2 attempt to calculate average
  - +1/2 return correct value
    - (Check for `int` division; must be `double` quotient)

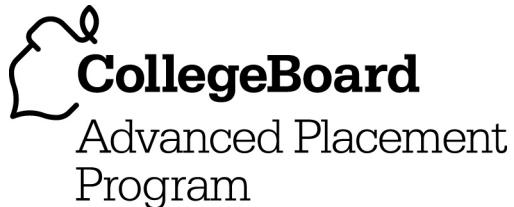
<b>Part B:</b>	<b>hasImproved</b>	<b>3 points</b>
----------------	--------------------	-----------------

- +1 loop over `scores`
  - +1/2 attempt (must reference `scores` in body)
  - +1/2 correct (will lose this if index out of bounds)
- +1 compare consecutive scores (in context of loop)
  - +1/2 attempt
  - +1/2 correct
- +1 return correct `boolean`
  - +1/2 categorize entire array as improved or not improved
    - (must be in context of comparing consecutive scores)
  - +1/2 correct value returned

<b>Part C:</b>	<b>finalAverage</b>	<b>3 points</b>
----------------	---------------------	-----------------

- +1 call `hasImproved()`
  - +1/2 attempt
  - +1/2 correct
- +1 return average of last half
  - +1/2 attempt to average half using `average`
  - +1/2 return correct average (only if improved)
- +1 return average of all
  - +1/2 attempt to average all using `average`
  - +1/2 return correct average (only if not improved)

Note: Reimplementing code rather than calling available methods results in score of 0 for the portion of part C related to the code reimplementation.



## **AP® Computer Science A 2004 Free-Response Questions**

**The materials included in these files are intended for noncommercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program®. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. This permission does not apply to any third-party copyrights contained herein. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here.**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,500 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit [www.collegeboard.com](http://www.collegeboard.com)

Copyright © 2004 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. PSAT/NMSQT is a registered trademark jointly owned by the College Entrance Examination Board and the National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

# **2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS**

## **COMPUTER SCIENCE A**

### **SECTION II**

**Time—1 hour and 45 minutes**

**Number of questions—4**

**Percent of total grade—50**

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

**Notes:**

- Assume that the classes listed in the Quick Reference found in the Appendix 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.

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

1. The following class WordList is designed to store and manipulate a list of words. The incomplete class declaration is shown below. You will be asked to implement two methods.

```
public class WordList
{
    private ArrayList myList; // contains Strings made up of letters

    // postcondition: returns the number of words in this WordList that
    //                 are exactly len letters long
    public int numWordsOfLength(int len)
    { /* to be implemented in part (a) */ }

    // postcondition: all words that are exactly len letters long
    //                 have been removed from this WordList, with the
    //                 order of the remaining words unchanged
    public void removeWordsOfLength(int len)
    { /* to be implemented in part (b) */ }

    // ... constructor and other methods not shown
}
```

- (a) Write the WordList method numWordsOfLength. Method numWordsOfLength returns the number of words in the WordList that are exactly len letters long. For example, assume that the instance variable myList of the WordList animals contains the following.

```
["cat", "mouse", "frog", "dog", "dog"]
```

The table below shows several sample calls to numWordsOfLength.

<u>Call</u>	<u>Result returned by call</u>
animals.numWordsOfLength(4)	1
animals.numWordsOfLength(3)	3
animals.numWordsOfLength(2)	0

Complete method numWordsOfLength below.

```
// postcondition: returns the number of words in this WordList that
//                 are exactly len letters long
public int numWordsOfLength(int len)
```

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (b) Write the WordList method `removeWordsOfLength`. Method `removeWordsOfLength` removes all words from the WordList that are exactly `len` letters long, leaving the order of the remaining words unchanged. For example, assume that the instance variable `myList` of the WordList `animals` contains the following.

```
["cat", "mouse", "frog", "dog", "dog"]
```

The table below shows a sequence of calls to the `removeWordsOfLength` method.

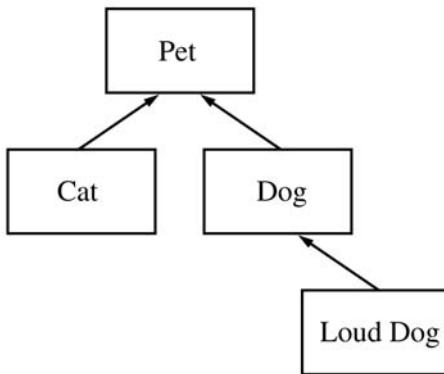
<u>Call</u>	<u>myList after the call</u>
<code>animals.removeWordsOfLength(4);</code>	<code>["cat", "mouse", "dog", "dog"]</code>
<code>animals.removeWordsOfLength(3);</code>	<code>["mouse"]</code>
<code>animals.removeWordsOfLength(2);</code>	<code>["mouse"]</code>

Complete method `removeWordsOfLength` below.

```
// postcondition: all words that are exactly len letters long  
//                  have been removed from this WordList, with the  
//                  order of the remaining words unchanged  
public void removeWordsOfLength(int len)
```

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

2. Consider the hierarchy of classes shown in the following diagram.



Note that a Cat “is-a” Pet, a Dog “is-a” Pet, and a LoudDog “is-a” Dog.

The class `Pet` is specified as an abstract class as shown in the following declaration. Each `Pet` has a name that is specified when it is constructed.

```
public abstract class Pet
{
    private String myName;

    public Pet(String name)
    {   myName = name;   }

    public String getName()
    {   return myName;   }

    public abstract String speak();
}
```

The subclass `Dog` has the partial class declaration shown below.

```
public class Dog extends Pet
{
    public Dog(String name)
    {   /* implementation not shown */   }

    public String speak()
    {   /* implementation not shown */   }
}
```

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

- (a) Given the class hierarchy shown above, write a complete class declaration for the class `Cat`, including implementations of its constructor and method(s). The `Cat` method `speak` returns "meow" when it is invoked.
- (b) Assume that class `Dog` has been declared as shown at the beginning of the question. If the `String dog-sound` is returned by the `Dog` method `speak`, then the `LoudDog` method `speak` returns a `String` containing `dog-sound` repeated two times.

Given the class hierarchy shown previously, write a complete class declaration for the class `LoudDog`, including implementations of its constructor and method(s).

- (c) Consider the following partial declaration of class `Kennel`.

```
public class Kennel
{
    private ArrayList petList;          // all elements are references
                                         // to Pet objects

    // postcondition: for each Pet in the kennel, its name followed
    //                  by the result of a call to its speak method
    //                  has been printed, one line per Pet
    public void allSpeak()
    { /* to be implemented in this part */ }

    // ... constructor and other methods not shown
}
```

Write the `Kennel` method `allSpeak`. For each `Pet` in the kennel, `allSpeak` prints a line with the name of the `Pet` followed by the result of a call to its `speak` method.

In writing `allSpeak`, you may use any of the methods defined for any of the classes specified for this problem. Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `allSpeak` below.

```
// postcondition: for each Pet in the kennel, its name followed
//                  by the result of a call to its speak method
//                  has been printed, one line per Pet
public void allSpeak()
```

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

4. The PR2004 is a robot that automatically gathers toys and other items scattered in a tiled hallway. A tiled hallway has a wall at each end and consists of a single row of tiles, each with some number of items to be gathered.

The PR2004 robot is initialized with a starting position and an array that contains the number of items on each tile. Initially the robot is facing right, meaning that it is facing toward higher-numbered tiles.

The PR2004 robot makes a sequence of moves until there are no items remaining on any tile. A move is defined as follows.

1. If there are any items on the current tile, then one item is removed.
2. If there are more items on the current tile, then the robot remains on the current tile facing the same direction.
3. If there are no more items on the current tile
  - a) if the robot can move forward, it advances to the next tile in the direction that it is facing;
  - b) otherwise, if the robot cannot move forward, it reverses direction and does not change position.

In the following example, the position and direction of the robot are indicated by "<" or ">" and the entries in the diagram indicate the number of items to be gathered on each tile. There are four tiles in this hallway. The starting state of the robot is illustrated in the following diagram.

Tile number:                    0        1        2        3  
Number of items: left wall → [ 1 | 1 | 2 | 2 ] ← right wall  
Robot position:                    >

The following sequence shows the configuration of the hallway and the robot after each move.

After move 1	After move 2	After move 3	After move 4
0    1    2    3	0    1    2    3	0    1    2    3	0    1    2    3
[ 1   0   2   2 ]	[ 1   0   1   2 ]	[ 1   0   0   2 ]	[ 1   0   0   1 ]
>	>	>	>

After move 5	After move 6	After move 7	After move 8
0    1    2    3	0    1    2    3	0    1    2    3	0    1    2    3
[ 1   0   0   0 ]	[ 1   0   0   0 ]	[ 1   0   0   0 ]	[ 1   0   0   0 ]
<	<	<	<

After move 9
[ 0   0   0   0 ]

After nine moves, the robot stops because the hall is clear.

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

The PR2004 is modeled by the class `Robot` as shown in the following declaration.

```
public class Robot
{
    private int[] hall;
    private int pos; // current position(tile number) of Robot
    private boolean facingRight; // true means this Robot is facing right

    // constructor not shown

    // postcondition: returns true if this Robot has a wall immediately in
    //                 front of it, so that it cannot move forward;
    //                 otherwise, returns false
    private boolean forwardMoveBlocked()
    { /* to be implemented in part (a) */ }

    // postcondition: one move has been made according to the
    //                 specifications above and the state of this
    //                 Robot has been updated
    private void move()
    { /* to be implemented in part (b) */ }

    // postcondition: no more items remain in the hallway;
    //                 returns the number of moves made
    public int clearHall()
    { /* to be implemented in part (c) */ }

    // postcondition: returns true if the hallway contains no items;
    //                 otherwise, returns false
    private boolean hallIsClear()
    { /* implementation not shown */ }
}
```

In the `Robot` class, the number of items on each tile in the hall is stored in the corresponding entry in the array `hall`. The current position is stored in the instance variable `pos`. The boolean instance variable `facingRight` is `true` if the Robot is facing to the right and is `false` otherwise.

- (a) Write the `Robot` method `forwardMoveBlocked`. Method `forwardMoveBlocked` returns `true` if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, `forwardMoveBlocked` returns `false`.

Complete method `forwardMoveBlocked` below.

```
// postcondition: returns true if this Robot has a wall immediately in
//                 front of it, so that it cannot move forward;
//                 otherwise, returns false
private boolean forwardMoveBlocked()
```

## 2004 AP® COMPUTER SCIENCE A FREE-RESPONSE QUESTIONS

(b) Write the Robot method move. Method move has the robot carry out one move as specified at the beginning of the question. The specification for a move is repeated here for your convenience.

1. If there are any items on the current tile, then one item is removed.
2. If there are more items on the current tile, then the robot remains on the current tile facing the same direction.
3. If there are no more items on the current tile
  - a) if the robot can move forward, it advances to the next tile in the direction that it is facing;
  - b) otherwise, if the robot cannot move forward, it reverses direction and does not change position.

In writing move, you may use any of the other methods in the Robot class. Assume these methods work as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method move below.

```
// postcondition: one move has been made according to the  
//                  specifications above and the state of this  
//                  Robot has been updated  
private void move()
```

(c) Write the Robot method clearHall. Method clearHall clears the hallway, repeatedly having this robot make a move until the hallway has no items, and returns the number of moves made.

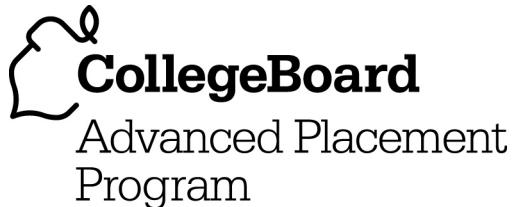
In the example at the beginning of this problem, clearHall would take the robot through the moves shown and return 9, leaving the robot in the state shown in the final diagram.

In writing clearHall, you may use any of the other methods in the Robot class. Assume these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method clearHall below.

```
// postcondition: no more items remain in the hallway;  
//                  returns the number of moves made  
public int clearHall()
```

**END OF EXAMINATION**



## **AP® Computer Science A 2004 Scoring Guidelines**

**The materials included in these files are intended for noncommercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program®. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. This permission does not apply to any third-party copyrights contained herein. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here.**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,500 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit [www.collegeboard.com](http://www.collegeboard.com)

Copyright © 2004 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

**AP® Computer Science A  
2004 SCORING GUIDELINES**

**Question 1**

<b>Part A:</b>	<b>numWordsOfLength</b>	<b>4 pts</b>
----------------	-------------------------	--------------

- +1/2 declare and initialize count to zero (could be an empty list, length = 0)  
(must show evidence that variable is used for counting or returned)
- +1 loop over myList
  - +1/2 attempt (must reference myList in body)
  - +1/2 correct
- +1/2 get String from myList (no deduction for missing downcast but local must be String)  
(lose this if array syntax used)
- +1 check length of String
  - +1/2 attempt (must be in context of loop)
  - +1/2 correct (array syntax is OK)
- +1/2 increment count (must be within context of length check)  
(lose this if count does not accumulate)
- +1/2 return correct count (after loop is completed)

<b>Part B:</b>	<b>removeWordsOfLength</b>	<b>5 pts</b>
----------------	----------------------------	--------------

- +2 loop over myList
  - +1 attempt (must reference myList in body)
  - +1 correct (must have attempt at removal, must not skip items)
- +1 get String from myList (no deduction for missing downcast, but local must be String)
  - +1/2 attempt (must be in context of loop, array syntax is OK)
  - +1/2 correct (no array syntax)
- +1 check length of String
  - +1/2 attempt (must be in context of loop)
  - +1/2 correct (array syntax is OK)
- +1 remove
  - +1/2 attempt (must call remove, must refer to myList or an index of an element in myList)
  - +1/2 correct (no array syntax)

**Usage:**

- 1/2 for WordList instead of myList
- 1/2 for returning a value in part B
- 1 for using this instead of myList, can lose in part A and again in part B (for max of -2)

**AP® Computer Science A  
2004 SCORING GUIDELINES**

**Question 2**

<b>Part A:</b>	<code>class Cat</code>	<b>2 pts</b>
----------------	------------------------	--------------

- +1/2 `public class Cat extends Pet`
- +1/2 Constructor correct (must call `super`)
- +1 speak method
  - +1/2 attempt (method header matches abstract method, OK if `abstract` left in)
  - +1/2 correct

<b>Part B:</b>	<code>class LoudDog</code>	<b>3 pts</b>
----------------	----------------------------	--------------

- +1/2 `public class LoudDog extends Dog`
- +1 Constructor correct (must call `super`)
- +1 1/2 speak method
  - +1 attempt (calls `super.speak()` *and* method header matches abstract method, OK if `abstract` left in)
  - +1/2 correct value returned

<b>Part C:</b>	<code>Kennel - allSpeak</code>	<b>4 pts</b>
----------------	--------------------------------	--------------

- +1 loop over `petList`
  - +1/2 attempt
  - +1/2 correct (must access `petList`)
- +1 1/2 get pet from `petList` (no deduction for missing downcast from `petList`)
  - +1/2 attempt
  - +1 correct (local variable must be type `Pet`)
- +1 1/2 print `p.getName()` and `p.speak()` for pet `p` (local variable not necessary)
  - +1/2 attempt (must have `xxx.getName()` or `xxx.speak()`, for some `xxx`)
  - +1 correct

Note: if done in-line with no local, no deduction for missing downcast.

---

Usage: -1/2 public instance variable

- 1 parent class name instead of `super`
- 1/2 `getName` is overridden (other than `super.getName`) in part (a) and/or part (b)

(No deduction for other additional methods or constructors.)

**AP® Computer Science A  
2004 SCORING GUIDELINES**

**Question 4**

<b>Part A:</b>	<code>forwardMoveBlocked</code>	<b>1 pt</b>
----------------	---------------------------------	-------------

- +1 return boolean
- +1/2 check a dir/pos pair
- +1/2 correct

<b>Part B:</b>	<code>move</code>	<b>5 pts</b>
----------------	-------------------	--------------

- +1 check for item(s) on current tile and remove one
  - +1/2 attempt on current tile (might try to remove all items)
  - +1/2 correct
- +1 1/2 check required conditions in context of attempt to move/turn (body of each check must refer to pos or facingRight)
  - +1 separate check for empty tile (e.g., not in ELSE)
  - +1/2 check `forwardMoveBlocked`
- +1 change direction (set direction to some value relative to current direction)
  - +1/2 toggle value
  - +1/2 if and only if originally blocked
- +1 1/2 move (set position to value(s) relative to current position)
  - +1/2 attempt 2 directions (change position, not value at position)
  - +1/2 move 1 tile in proper direction
  - +1/2 if and only if originally not blocked

<b>Part C:</b>	<code>clearHall</code>	<b>3 pts</b>
----------------	------------------------	--------------

- +1/2 declare and initialize counter (must have some extra context relevant to counting)
- +1 loop until done
  - +1/2 call to `hallIsClear` in loop
  - +1/2 correct
- +1 robot action (in context of a loop)
  - +1/2 call `move`
  - +1/2 correctly determine number of times `move` is called
- +1/2 always return number of times `move` is called (no credit for returning 0 with no call to `move` in code)

## APPENDIX C — SAMPLE SEARCH AND SORT ALGORITHMS

### Sequential Search

The Sequential Search Algorithm below finds the index of a value in an array of integers as follows:

1. Traverse `elements` until `target` is located, or the end of `elements` is reached.
2. If `target` is located, return the index of `target` in `elements`; Otherwise return `-1`.

```
/**  
 * Finds the index of a value in an array of integers.  
 *  
 * @param elements an array containing the items to be searched.  
 * @param target the item to be found in elements.  
 * @return an index of target in elements if found; -1 otherwise.  
 */  
public static int sequentialSearch(int[] elements, int target)  
{  
    for (int j = 0; j < elements.length; j++)  
    {  
        if (elements[j] == target)  
        {  
            return j;  
        }  
    }  
  
    return -1;  
}
```

## Binary Search

The Binary Search Algorithm below finds the index of a value in an array of integers sorted in ascending order as follows:

1. Set `left` and `right` to the minimum and maximum indexes of `elements` respectively.
2. Loop until `target` is found, or `target` is determined not to be in `elements` by doing the following for each iteration:
  - a. Set `middle` to the index of the middle item in `elements[left] ... elements[right]` inclusive.
  - b. If `target` would have to be in `elements[left] ... elements[middle - 1]` inclusive, then set `right` to the maximum index for that range.
  - c. Otherwise, if `target` would have to be in `elements[middle + 1] ... elements[right]` inclusive, then set `left` to the minimum index for that range.
  - d. Otherwise, return `middle` because `target == elements[middle]`.
3. Return `-1` if `target` is not contained in `elements`.

```

/**
 * Find the index of a value in an array of integers sorted in ascending order.
 *
 * @param elements an array containing the items to be searched.
 *      Precondition: items in elements are sorted in ascending order.
 * @param target the item to be found in elements.
 * @return an index of target in elements if target found;
 *         -1 otherwise.
 */
public static int binarySearch(int[] elements, int target)
{
    int left = 0;
    int right = elements.length - 1;

    while (left <= right)
    {
        int middle = (left + right) / 2;
        if (target < elements[middle])
        {
            right = middle - 1;
        }
        else if (target > elements[middle])
        {
            left = middle + 1;
        }
        else
        {
            return middle;
        }
    }

    return -1;
}

```

## Selection Sort

The Selection Sort Algorithm below sorts an array of integers into ascending order as follows:

1. Loop from `j = 0` to `j = elements.length-2`, inclusive, completing `elements.length-1` passes.
2. In each pass, swap the item at index `j` with the minimum item in the rest of the array (`elements[j+1]` through `elements[elements.length-1]`).

At the end of each pass, items in `elements[0]` through `elements[j]` are in ascending order and each item in this sorted portion is at its final position in the array

```
/**  
 * Sort an array of integers into ascending order.  
 *  
 * @param elements an array containing the items to be sorted.  
 *  
 * Postcondition: elements contains its original items and items in elements  
 * are sorted in ascending order.  
 */  
public static void selectionSort(int[] elements)  
{  
    for (int j = 0; j < elements.length - 1; j++)  
    {  
        int minIndex = j;  
        for (int k = j + 1; k < elements.length; k++)  
        {  
            if (elements[k] < elements[minIndex])  
            {  
                minIndex = k;  
            }  
        }  
  
        int temp = elements[j];  
        elements[j] = elements[minIndex];  
        elements[minIndex] = temp;  
    }  
}
```

## Insertion Sort

The Insertion Sort Algorithm below sorts an array of integers into ascending order as follows:

1. Loop from `j = 1` to `j = elements.length-1` inclusive, completing `elements.length-1` passes.
2. In each pass, move the item at index `j` to its proper position in `elements[0]` to `elements[j]`:
  - a. Copy item at index `j` to `temp`, creating a “vacant” element at index `j` (denoted by `possibleIndex`).
  - b. Loop until the proper position to maintain ascending order is found for `temp`.
  - c. In each inner loop iteration, move the “vacant” element one position lower in the array.
3. Copy `temp` into the identified correct position (at `possibleIndex`).

At the end of each pass, items at `elements[0]` through `elements[j]` are in ascending order.

```
/**  
 * Sort an array of integers into ascending order.  
 *  
 * @param elements an array containing the items to be sorted.  
 *  
 * Postcondition: elements contains its original items and items in elements  
 * are sorted in ascending order.  
 */  
public static void insertionSort(int[] elements)  
{  
    for (int j = 1; j < elements.length; j++)  
    {  
        int temp = elements[j];  
        int possibleIndex = j;  
        while (possibleIndex > 0 && temp < elements[possibleIndex - 1])  
        {  
            elements[possibleIndex] = elements[possibleIndex - 1];  
            possibleIndex--;  
        }  
        elements[possibleIndex] = temp;  
    }  
}
```

## Merge Sort

The Merge Sort Algorithm below sorts an array of integers into ascending order as follows:

### `mergeSort`

This top-level method creates the necessary temporary array and calls the `mergeSortHelper` recursive helper method.

### `mergeSortHelper`

This recursive helper method uses the Merge Sort Algorithm to sort `elements[from] ... elements[to]` inclusive into ascending order:

1. If there is more than one item in this range,
  - a. divide the items into two adjacent parts, and
  - b. call `mergeSortHelper` to recursively sort each part, and
  - c. call the `merge` helper method to merge the two parts into sorted order.
2. Otherwise, exit because these items are sorted.

### `merge`

This helper method merges two adjacent array parts, each of which has been sorted into ascending order, into one array part that is sorted into ascending order:

1. As long as both array parts have at least one item that hasn't been copied, compare the first un-copied item in each part and copy the minimal item to the next position in `temp`.
2. Copy any remaining items of the first part to `temp`.
3. Copy any remaining items of the second part to `temp`.
4. Copy the items from `temp[from] ... temp[to]` inclusive to the respective locations in `elements`.

```
/**  
 * Sort an array of integers into ascending order.  
 *  
 * @param elements an array containing the items to be sorted.  
 *  
 * Postcondition: elements contains its original items and items in elements  
 * are sorted in ascending order.  
 */  
public static void mergeSort(int[] elements)  
{  
    int n = elements.length;  
    int[] temp = new int[n];  
    mergeSortHelper(elements, 0, n - 1, temp);  
}
```

```

/**
 * Sorts elements[from] ... elements[to] inclusive into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 * @param from the beginning index of the items in elements to be sorted.
 * @param to the ending index of the items in elements to be sorted.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition:
 *      (elements.length == 0 or
 *       0 <= from <= to <= elements.length) and
 *       elements.length == temp.length
 * Postcondition: elements contains its original items and the items in elements
 * [from] ... <= elements[to] are sorted in ascending order.
 */
private static void mergeSortHelper(int[] elements,
                                    int from, int to, int[] temp)
{
    if (from < to)
    {
        int middle = (from + to) / 2;
        mergeSortHelper(elements, from, middle, temp);
        mergeSortHelper(elements, middle + 1, to, temp);
        merge(elements, from, middle, to, temp);
    }
}

```

```

/**
 * Merges two adjacent array parts, each of which has been sorted into ascending
 * order, into one array part that is sorted into ascending order.
 *
 * @param elements an array containing the parts to be merged.
 * @param from the beginning index in elements of the first part.
 * @param mid the ending index in elements of the first part.
 *           mid+1 is the beginning index in elements of the second part.
 * @param to the ending index in elements of the second part.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition: 0 <= from <= mid <= to <= elements.length and
 * elements[from] ... <= elements[mid] are sorted in ascending order and
 * elements[mid + 1] ... <= elements[to] are sorted in ascending order and
 * elements.length == temp.length
 * Postcondition: elements contains its original items and
 * elements[from] ... <= elements[to] are sorted in ascending order and
 * elements[0] ... elements[from - 1] are in original order and
 * elements[to + 1] ... elements[elements.length - 1] are in original order.
 */
private static void merge(int[] elements,
                        int from, int mid, int to, int[] temp)
{
    int i = from;
    int j = mid + 1;
    int k = from;

    while (i <= mid && j <= to)
    {
        if (elements[i] < elements[j])
        {
            temp[k] = elements[i];
            i++;
        }
        else
        {
            temp[k] = elements[j];
            j++;
        }
        k++;
    }
}

```

```
while (i <= mid)
{
    temp[k] = elements[i];
    i++;
    k++;
}

while (j <= to)
{
    temp[k] = elements[j];
    j++;
    k++;
}

for (k = from; k <= to; k++)
{
    elements[k] = temp[k];
}
}
```