AP® Computer Science A Exam

SECTION II: Free Response, Questions

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.

At a Glance

Total Time
1 hour and 30 minutes
Number of Questions
4
Percent of Total Score
50%
Writing Instrument
Pencil
Electronic Device
None allowed
Weight
The questions are weighted equally.

Instructions

The questions for Section II are printed in this booklet. You may use the pages in this booklet to organize your answers and for scratch work, but you must write your answers in the blank space provided for each question.

The Java Quick Reference is located inside the front cover of this booklet.

Write your answer to each question in the blank space provided. Begin your response to each question at the top of a new page and completely fill in the circle at the top of each page that corresponds to the question you are answering.

All program segments must be written in Java. Show all your work. Credit for partial solutions will be given. Write clearly and legibly. Erased or crossed-out work will not be scored.

Manage your time carefully. Do not spend too much time on any one question. You may proceed freely from one question to the next. You may review your responses if you finish before the end of the exam is announced.

Java Quick Reference

Accessible methods from the Java library that may be included in the exam

Class Constructors and Methods	Explanation
String Class	
`String(String str)`	Constructs a new `String` object that represents the same sequence of characters as `str`
`int length()`	Returns the number of characters in a `String` object
`String substring(int from, int to)`	Returns the substring beginning at index `from` and ending at index `to - 1`
`String substring(int from)`	Returns `substring(from, length())`
`int indexOf(String str)`	Returns the index of the first occurrence of `str`; returns `-1` if not found
`boolean equals(String other)`	Returns `true` if `this` is equal to `other`; returns `false` otherwise
`int compareTo(String other)`	Returns a value `<0` if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value `>0` if `this` is greater than `other`
Integer Class	
`Integer(int value)`	Constructs a new `Integer` object that represents the specified `int` value
`Integer.MIN_VALUE`	The minimum value represented by an `int` or `Integer`
`Integer.MAX_VALUE`	The maximum value represented by an `int` or `Integer`
`int intValue()`	Returns the value of this `Integer` as an `int`
Double Class	
`Double(double value)`	Constructs a new `Double` object that represents the specified `double` value
`double doubleValue()`	Returns the value of this `Double` as a `double`
Math Class	
`static int abs(int x)`	Returns the absolute value of an `int` value
`static double abs(double x)`	Returns the absolute value of a `double` value
`static double pow(double base, double exponent)`	Returns the value of the first parameter raised to the power of the second parameter
`static double sqrt(double x)`	Returns the positive square root of a `double` value
`static double random()`	Returns a `double` value greater than or equal to `0.0` and less than `1.0`
ArrayList Class	
`int size()`	Returns the number of elements in the list
`boolean add(E obj)`	Appends `obj` to end of list; returns `true`
`void add(int index, E obj)`	Inserts `obj` at position `index (0 <= index <= size)`, moving elements at position `index` and higher to the right (adds 1 to their indices) and adds 1 to size
`E get(int index)`	Returns the element at position `index` in the list
`E set(int index, E obj)`	Replaces the element at position `index` with `obj`; returns the element formerly at position `index`
`E remove(int index)`	Removes element from position `index`, moving elements at position `index + 1` and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position `index`
Object Class	
`boolean equals(Object other)`	
`String toString()`	

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 Questions booklet, but no credit will be given for anything written in this booklet. You will only earn credit for what you write in the Free Response booklet.

Notes:

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

GO ON TO THE NEXT PAGE.

1. A mathematical sequence is an ordered list of numbers. This question involves a sequence called a *hailstone sequence*. If n is the value of a term in the sequence, then the following rules are used to find the next term, if one exists.

   - If n is 1, the sequence terminates.
   - If n is even, then the next term is $\frac{n}{2}$.
   - If n is odd, then the next term is $3n + 1$.

   For this question, assume that when the rules are applied, the sequence will eventually terminate with the term $n = 1$.

   The following are examples of hailstone sequences.

   Example 1: 5, 16, 8, 4, 2, 1

   - The first term is 5, so the second term is $5 * 3 + 1 = 16$.
   - The second term is 16, so the third term is $\frac{16}{2} = 8$.
   - The third term is 8, so the fourth term is $\frac{8}{2} = 4$.
   - The fourth term is 4, so the fifth term is $\frac{4}{2} = 2$.
   - The fifth term is 2, so the sixth term is $\frac{2}{2} = 1$.
   - Since the sixth term is 1, the sequence terminates.

   Example 2: 8, 4, 2, 1

   - The first term is 8, so the second term is $\frac{8}{2} = 4$.
   - The second term is 4, so the third term is $\frac{4}{2} = 2$.
   - The third term is 2, so the fourth term is $\frac{2}{2} = 1$.
   - Since the fourth term is 1, the sequence terminates.

GO ON TO THE NEXT PAGE.

The Hailstone class, shown below, is used to represent a hailstone sequence. You will write three methods in the Hailstone class.

```
public class Hailstone
{
   /** Returns the length of a hailstone sequence that starts with n,
    *  as described in part (a).
    *  Precondition: n > 0
    */
   public static int hailstoneLength(int n)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the hailstone sequence that starts with n is considered long
    *  and false otherwise, as described in part (b).
    *  Precondition: n > 0
    */
   public static boolean isLongSeq(int n)
   {  /* to be implemented in part (b) */  }

   /** Returns the proportion of the first n hailstone sequences that are considered long,
    *  as described in part (c).
    *  Precondition: n > 0
    */
   public static double propLong(int n)
   {  /* to be implemented in part (c) */  }

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

GO ON TO THE NEXT PAGE.

(a) The length of a hailstone sequence is the number of terms it contains. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) has a length of 6 and the hailstone sequence in example 2 (8, 4, 2, 1) has a length of 4.

Write the method `hailstoneLength(int n)`, which returns the length of the hailstone sequence that starts with `n`.

```
/** Returns the length of a hailstone sequence that starts with  n,  as described in part (a).
 *  Precondition: n > 0
 */
public static int hailstoneLength(int n)
```

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 Hailstone

public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

GO ON TO THE NEXT PAGE.

(b) A hailstone sequence is considered long if its length is greater than its starting value. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) is considered long because its length (6) is greater than its starting value (5). The hailstone sequence in example 2 (8, 4, 2, 1) is not considered long because its length (4) is less than or equal to its starting value (8).

Write the method `isLongSeq(int n)`, which returns `true` if the hailstone sequence starting with `n` is considered long and returns `false` otherwise. Assume that `hailstoneLength` works as intended, regardless of what you wrote in part (a). You must use `hailstoneLength` appropriately to receive full credit.

```
/** Returns true if the hailstone sequence that starts with n is considered long
 *  and false otherwise, as described in part (b).
 *  Precondition: n > 0
 */
public static boolean isLongSeq(int n)
```

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.

(c) The method `propLong(int n)` returns the proportion of long hailstone sequences with starting values between 1 and `n`, inclusive.

Consider the following table, which provides data about the hailstone sequences with starting values between 1 and 10, inclusive.

Starting Value	Terms in the Sequence	Length of the Sequence	Long?
1	1	1	No
2	2, 1	2	No
3	3, 10, 5, 16, 8, 4, 2, 1	8	Yes
4	4, 2, 1	3	No
5	5, 16, 8, 4, 2, 1	6	Yes
6	6, 3, 10, 5, 16, 8, 4, 2, 1	9	Yes
7	7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	17	Yes
8	8, 4, 2, 1	4	No
9	9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	20	Yes
10	10, 5, 16, 8, 4, 2, 1	7	No

The method call `Hailstone.propLong(10)` returns `0.5`, since 5 of the 10 hailstone sequences shown in the table are considered long.

Write the `propLong` method. Assume that `hailstoneLength` and `isLongSeq` work as intended, regardless of what you wrote in parts (a) and (b). You must use `isLongSeq` appropriately to receive full credit.

```
/** Returns the proportion of the first n hailstone sequences that are considered long,
 *  as described in part (c).
 *  Precondition: n > 0
 */
public static double propLong(int n)
```

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 Hailstone

public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

GO ON TO THE NEXT PAGE.

NO TEST MATERIAL ON THIS PAGE

GO ON TO THE NEXT PAGE.

2. This question involves the creation and use of a spinner to generate random numbers in a game. A `GameSpinner` object represents a spinner with a given number of sectors, all equal in size. The `GameSpinner` class supports the following behaviors.

- Creating a new spinner with a specified number of sectors
- Spinning a spinner and reporting the result
- Reporting the length of the *current run*, the number of consecutive spins that are the same as the most recent spin

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

Statements	**Value Returned (blank if no value returned)**	**Comment**
`GameSpinner g = new GameSpinner(4);`		Creates a new spinner with four sectors
`g.currentRun();`	`0`	Returns the length of the current run. The length of the current run is initially `0` because no spins have occurred.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because there has been one spin of `3` so far.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`2`	The length of the current run is `2` because there have been two `3`s in a row.
`g.spin();`	`4`	Returns a random integer between `1` and `4`, inclusive. In this case, `4` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because the spin of `4` is different from the value of the spin in the previous run of two `3`s.
`g.spin();`	`3`	Returns a random integer between `1` and `4`, inclusive. In this case, `3` is returned.
`g.currentRun();`	`1`	The length of the current run is `1` because the spin of `3` is different from the value of the spin in the previous run of one `4`.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.spin();`	`1`	Returns a random integer between `1` and `4`, inclusive. In this case, `1` is returned.
`g.currentRun();`	`3`	The length of the current run is `3` because there have been three consecutive `1`s since the previous run of one `3`.

GO ON TO THE NEXT PAGE.

Write the complete `GameSpinner` class. Your implementation must meet all specifications and conform to the example.

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

GO ON TO THE NEXT PAGE.

3. A student plans to analyze product reviews found on a Web site by looking for keywords in posted reviews. The `ProductReview` class, shown below, is used to represent a single review. A product review consists of a product name and a review of that product.

```
public class ProductReview
{
   private String name;
   private String review;

   /** Constructs a ProductReview object and initializes the instance variables. */
   public ProductReview(String pName, String pReview)
   {
      name = pName;
      review = pReview;
   }

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

   /** Returns the review of the product. */
   public String getReview()
   {  return review; }
}
```

The `ReviewCollector` class, shown below, is used to represent a collection of reviews to be analyzed.

```
public class ReviewCollector
{
   private ArrayList<ProductReview> reviewList;
   private ArrayList<String> productList;

   /** Constructs a ReviewCollector object and initializes the instance variables. */
   public ReviewCollector()
   {
      reviewList = new ArrayList<ProductReview>();
      productList = new ArrayList<String>();
   }

   /** Adds a new review to the collection of reviews, as described in part (a). */
   public void addReview(ProductReview prodReview)
   {  /* to be implemented in part (a) */  }

   /** Returns the number of good reviews for a given product name, as described in part (b). */
   public int getNumGoodReviews(String prodName)
   {  /* to be implemented in part (b) */  }

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

GO ON TO THE NEXT PAGE.

(a) Write the `addReview` method, which adds a single product review, represented by a `ProductReview` object, to the `ReviewCollector` object. The `addReview` method does the following when it adds a product review.

- The `ProductReview` object is added to the `reviewList` instance variable.
- The product name from the `ProductReview` object is added to the `productList` instance variable if the product name is not already found in `productList`.

Elements may be added to `reviewList` and `productList` in any order.

Complete method `addReview`.

```
/** Adds a new review to the collection of reviews, as described in part (a). */
public void addReview(ProductReview prodReview)
```

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.

(b) Write the `getNumGoodReviews` method, which returns the number of *good* reviews for a given product name. A review is considered good if it contains the string `"best"` (all lowercase). If there are no reviews with a matching product name, the method returns `0`. Note that a review that contains `"BEST"` or `"Best"` is not considered a good review (since not all the letters of `"best"` are lowercase), but a review that contains `"asbestos"` is considered a good review (since all the letters of `"best"` are lowercase).

Complete method `getNumGoodReviews`.

```
/** Returns the number of good reviews for a given product name, as described in part (b). */
public int getNumGoodReviews(String prodName)
```

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 ProductReview

private String name
private String review

public ProductReview(String pName, String pReview)
public String getName()
public String getReview()

public class ReviewCollector

private ArrayList<ProductReview> reviewList
private ArrayList<String> productList

public ReviewCollector()
public void addReview(ProductReview prodReview)
public int getNumGoodReviews(String prodName)
```

GO ON TO THE NEXT PAGE.

NO TEST MATERIAL ON THIS PAGE

GO ON TO THE NEXT PAGE.

4. A theater contains rows of seats with the same number of seats in each row. Some rows contain tier 1 seats, and the remaining rows contain tier 2 seats. Tier 1 seats are closer to the stage and are more desirable. All seats in a row share the same tier.

The `Seat` class, shown below, represents seats in the theater. The `boolean` instance variable `available` is `false` if a ticket for the seat has been sold (the seat is no longer available). The `int` instance variable `tier` indicates whether the seat is a tier 1 or tier 2 seat.

```
public class Seat
{
   private boolean available;
   private int tier;

   public Seat(boolean isAvail, int tierNum)
   {
      available = isAvail;
      tier = tierNum;
   }

   public boolean isAvailable()
   {  return available;  }

   public int getTier()
   {  return tier;  }

   public void setAvailability(boolean isAvail)
   {  available = isAvail;  }
}
```

GO ON TO THE NEXT PAGE.

The `Theater` class represents a theater of seats. The number of seats per row and the number of tier 1 and tier 2 rows are determined by the parameters of the `Theater` constructor. Row `0` of the `theaterSeats` array represents the row closest to the stage.

```
public class Theater
{
   private Seat[][] theaterSeats;

   /** Constructs a Theater object, as described in part (a).
    *  Precondition: seatsPerRow > 0; tier1Rows > 0; tier2Rows >= 0
    */
   public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
   {  /* to be implemented in part (a) */  }

   /** Returns true if a seat holder was reassigned from the seat at fromRow, fromCol
    *   to the seat at toRow, toCol; otherwise it returns false, as described in part (b).
    *  Precondition: fromRow, fromCol, toRow, and toCol represent valid row and
    *                column positions in the theater.
    *                The seat at fromRow, fromCol is not available.
    */
   public boolean reassignSeat(int fromRow, int fromCol,
                               int toRow, int toCol)
   {  /* to be implemented in part (b) */  }
}
```

GO ON TO THE NEXT PAGE.

(a) Write the constructor for the `Theater` class. The constructor takes three `int` parameters, representing the number of seats per row, the number of tier 1 rows, and the number of tier 2 rows, respectively. The constructor initializes the `theaterSeats` instance variable so that it has the given number of seats per row and the given number of tier 1 and tier 2 rows and all seats are available and have the appropriate tier designation.

Row `0` of the `theaterSeats` array represents the row closest to the stage. All tier 1 seats are closer to the stage than tier 2 seats.

Complete the `Theater` constructor.

```
/** Constructs a Theater object, as described in part (a).
 *  Precondition: seatsPerRow > 0; tier1Rows > 0; tier2Rows >= 0
 */
public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
```

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 Seat

private boolean available
private int tier

public Seat(boolean isAvail, int tierNum)
public boolean isAvailable()
public int getTier()
public void setAvailability(boolean isAvail)

public class Theater

private Seat[][] theaterSeats

public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
public boolean reassignSeat(int fromRow, int fromCol,
                            int toRow, int toCol)
```

GO ON TO THE NEXT PAGE.

(b) Write the `reassignSeat` method, which attempts to move a person from a source seat to a destination seat. The reassignment can be made if the destination seat is available and has the same or greater tier than the source seat (that is, it is equally or less desirable). For example, a person in a tier 1 seat can be moved to a different tier 1 seat or to a tier 2 seat, but a person in a tier 2 seat can only be moved to a different tier 2 seat.

The `reassignSeat` method has four `int` parameters representing the row and column indexes of the source ("from") and destination ("to") seats. If the reassignment is possible, the source seat becomes available, the destination seat becomes unavailable, and the method returns `true`. If the seat reassignment is not possible, no changes are made to either seat and the method returns `false`. Assume that the source seat is occupied when the method is called.

Complete method `reassignSeat`.

```
/** Returns true if a seat holder was reassigned from the seat at fromRow, fromCol
 *  to the seat at toRow, toCol; otherwise it returns false, as described in part (b).
 *  Precondition: fromRow, fromCol, toRow, and toCol represent valid row and
 *                column positions in the theater.
 *                The seat at fromRow, fromCol is not available.
 */
public boolean reassignSeat(int fromRow, int fromCol,
                            int toRow, int toCol)
```

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.

STOP

END OF EXAM