Complete method `makeAppointment`. Assume that `findFreeBlock` works as intended, regardless of what you wrote in part (a). You must use `findFreeBlock` and `reserveBlock` appropriately in order to receive full credit.

```
/**
 *   Searches periods from startPeriod to endPeriod, inclusive, for a block
 *   of duration free minutes, as described in part (b). If such a block is found,
 *   calls reserveBlock to reserve the block of minutes and returns true; otherwise
 *   returns false.
 *   Preconditions: 1 <= startPeriod <= endPeriod <= 8; 1 <= duration <= 60
 */
public boolean makeAppointment(int startPeriod, int endPeriod,
                               int duration)
```

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

Class information for this question

```
public class AppointmentBook

private boolean isMinuteFree(int period, int minute)
private void reserveBlock(int period, int startMinute,
   int duration)
public int findFreeBlock(int period, int duration)
public boolean makeAppointment(int startPeriod, int endPeriod,
   int duration)
```

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

2. This question involves methods that distribute text across lines of an electronic sign. The electronic sign and the text to be displayed on it are represented by the `Sign` class. You will write the complete `Sign` class, which contains a constructor and two methods.

The `Sign` class constructor has two parameters. The first parameter is a `String` that contains the message to be displayed on the sign. The second parameter is an `int` that contains the *width* of each line of the sign. The width is the positive maximum number of characters that can be displayed on a single line of the sign.

A sign contains as many lines as are necessary to display the entire message. The message is split among the lines of the sign without regard to spaces or punctuation. Only the last line of the sign may contain fewer characters than the width indicated by the constructor parameter.

The following are examples of a message displayed on signs of different widths. Assume that in each example, the sign is declared with the width specified in the first column of the table and with the message `"Everything on sale, please come in"`, which contains `34` characters.

Width of the Sign	Sign Display

15	**`Everything on s`**
	`ale, please com`
	`e in`

17	**`Everything on sal`**
	`e, please come in`

40	**`Everything on sale, please come in`**

In addition to the constructor, the `Sign` class contains two methods.

The `numberOfLines` method returns an `int` representing the number of lines needed to display the text on the sign. In the previous examples, `numberOfLines` would return `3`, `2`, and `1`, respectively, for the sign widths shown in the table.

The `getLines` method returns a `String` containing the message broken into lines separated by semicolons (`;`) or returns `null` if the message is the empty string. The constructor parameter that contains the message to be displayed will not include any semicolons. As an example, in the first row of the preceding table, `getLines` would return `"Everything on s;ale, please com;e in"`. No semicolon should appear at the end of the `String` returned by `getLines`.

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

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

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

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

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

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

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

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

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

99.1	142.0	85.0	85.1	84.6	94.3	124.9	98.0	101.0	102.5

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

99.1	142.0	85.0	85.1	84.6	94.3	124.9	98.0	101.0	102.5

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

99.1	85.0	85.1	94.3	98.0	101.0	102.5

Complete method `cleanData`.

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

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

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

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

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

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

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

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

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

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

Complete method `longestHeatWave`.

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

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

Class information for this question

```
public class WeatherData

private ArrayList<Double> temperatures

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

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

4. This question involves pieces of candy in a box. The `Candy` class represents a single piece of candy.

```
public class Candy
{
   /** Returns a String representing the flavor of this piece of candy */
   public String getFlavor()
   {  /* implementation not shown */  }

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

The `BoxOfCandy` class represents a candy box where the candy is arranged in a rectangular grid. The instance variable of the class, `box`, is a rectangular two-dimensional array of `Candy` objects. A location in the candy box may contain a piece of candy or may be empty. A piece of candy is represented by a `Candy` object. An empty location is represented by `null`.

You will write two methods of the `BoxOfCandy` class.

```
public class BoxOfCandy
{
   /** box contains at least one row and is initialized in the constructor. */
   private Candy[][] box;

   /**
    *  Moves one piece of candy in column col, if necessary and possible, so that the box
    *  element in row 0 of column col contains a piece of candy, as described in part (a).
    *  Returns false if there is no piece of candy in column col and returns true otherwise.
    *  Precondition: col is a valid column index in box.
    */
   public boolean moveCandyToFirstRow(int col)
   {  /* to be implemented in part (a) */  }

   /**
    *  Removes from box and returns a piece of candy with flavor specified by the parameter, or
    *  returns null if no such piece is found, as described in part (b)
    */
   public Candy removeNextByFlavor(String flavor)
   {  /* to be implemented in part (b) */  }

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

GO ON TO THE NEXT PAGE.

© 2023 College Board.
Visit College Board on the web: collegeboard.org.

Question 3: Array / ArrayList — 9 points

Canonical solution

(a) **4 points**

```
public void cleanData(double lower, double upper)
{
   for (int i = temperatures.size() - 1; i >= 0; i--)
   {
      double temp = temperatures.get(i);
      if (temp < lower || temp > upper)
      {
         temperatures.remove(i);
      }
   }
}
```

(b) **5 points**

```
public int longestHeatWave(double threshold)
{
   int waveLength = 0;
   int maxWaveLength = 0;

   for (double temp : temperatures)
   {
      if (temp > threshold)
      {
         waveLength++;
         if (waveLength > maxWaveLength)
         {
            maxWaveLength = waveLength;
         }
      }
      else
      {
         waveLength = 0;
      }
   }
   return maxWaveLength;
}
```

© 2023 College Board

(a) `cleanData`

	Scoring Criteria	Decision Rules	
1	Traverses `temperatures` (*no bounds errors*)	Responses **can** still earn the point even if they • do a forward traversal of the list • skip a value because removal from the list is handled incorrectly • use an enhanced `for` loop, as long as the list element is used in the body of the loop Responses **will not** earn the point if they • fail to ever access an element of `temperatures` in the loop • access the elements of `temperatures` incorrectly	**1 point**
2	Determines whether an element of temperature list should be removed, using `lower` and `upper`	Responses **can** still earn the point even if they • access elements of temperature list incorrectly Responses **will not** earn the point if they • apply incorrect comparison (`<` vs `<=`) or logic (`||` vs `&&`) to identify elements of list for removal	**1 point**
3	Calls `remove` (or equivalent) on temperature list with an appropriate parameter	Responses **can** still earn the point even if they • add the element to a new `ArrayList` that is not declared, is declared incorrectly, or is not assigned to the instance variable, as long as the order of elements is maintained Responses **will not** earn the point if they • call `remove` or `add` incorrectly	**1 point**
4	Removes all and only identified elements of temperature list (*algorithm*)	Responses **can** still earn the point even if they • call `remove` incorrectly • access the elements of temperature list incorrectly Responses **will not** earn the point if they • add elements to a new `ArrayList` that is not declared, is declared incorrectly, or is not assigned to the instance variable • skip a temperature list element in the traversal because removal is not handled correctly	**1 point**
		Total for part (a)	**4 points**

(b) `longestHeatWave`

	Scoring Criteria	Decision Rules	
5	Traverses `temperatures` (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `temperatures` in the loop • access the elements of `temperatures` incorrectly	**1 point**
6	Compares an element of temperature list to `threshold` (*in the context of a loop*)	Responses **can** still earn the point even if they • always compare `threshold` to the same list element Responses **will not** earn the point if they • apply incorrect comparison (`>` vs `>=`) to identify heat wave days	**1 point**
7	Initializes and increments the length of a heat wave (*in the context of a loop or condition*)	Responses **can** still earn the point even if they • fail to reset the length of the current heat wave when the heat wave ends	**1 point**
8	Determines the length of at least one heat wave (*algorithm*)	Responses **will not** earn the point if they • fail to reset the length of the current heat wave when the heat wave ends	**1 point**
9	Identifies the longest heat wave and returns its length (*algorithm*)		**1 point**
		Total for part (b)	**5 points**
	Question-specific penalties		
	None		
		Total for question 3	**9 points**

Applying the Scoring Criteria

Apply the question scoring criteria 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. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
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., valid 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 (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `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 `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- 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 "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*