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.

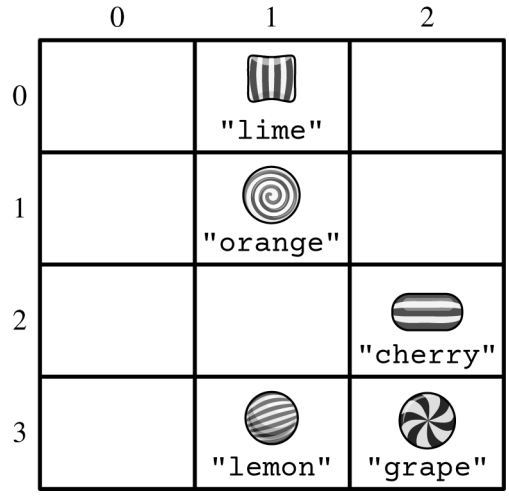
```
public class BoxOfCandy
{
    /** box contains at least one row and is initialized in the constructor. */
    private Candy[][] box;

    /**
     *  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.

(a) Write the `moveCandyToFirstRow` method, which attempts to ensure that the `box` element at row `0` and column `col` contains a piece of candy, using the following steps.

- If the element at row `0` and column `col` already contains a piece of candy, then `box` is unchanged and the method returns `true`.
- If the element at row `0` and column `col` does not contain a piece of candy, then the method searches the remaining rows of column `col` for a piece of candy. If a piece of candy can be found in column `col`, it is moved to row `0`, its previous location is set to `null`, and the method returns `true`; otherwise, the method returns `false`.

In the following example, the grid represents the contents of `box`. An empty square in the grid is `null` in `box`. A non-empty square in the grid represents a `box` element that contains a `Candy` object. The string in the square of the grid indicates the flavor of the piece of candy.

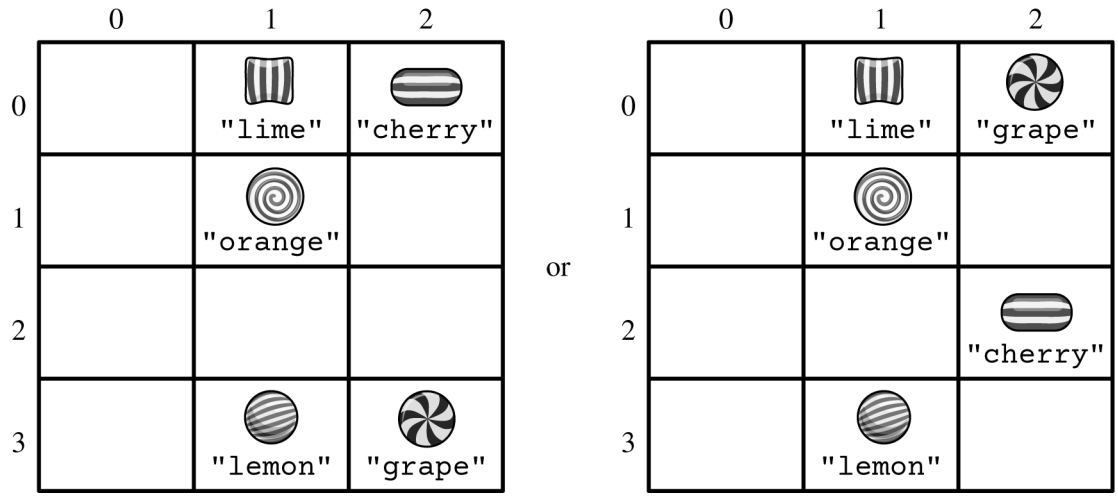


The method call `moveCandyToFirstRow(0)` returns `false` because the `box` element at row `0` and column `0` does not contain a piece of candy and there are no pieces of candy in column `0` that can be moved to row `0`. The contents of `box` are unchanged.

The method call `moveCandyToFirstRow(1)` returns `true` because the `box` element at row `0` and column `1` already contains a piece of candy. The contents of `box` are unchanged.

GO ON TO THE NEXT PAGE.

The method call `moveCandyToFirstRow(2)` moves one of the two pieces of candy in column `2` to row `0` of column `2`, sets the previous location of the piece of candy that was moved to `null`, and returns `true`. The new contents of `box` could be either of the following.



Complete the `moveCandyToFirstRow` method.

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

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

Class information for this question

```
public class Candy

public String getFlavor()

public class BoxOfCandy

private Candy[][] box

public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

GO ON TO THE NEXT PAGE.

(b) Write the `removeNextByFlavor` method, which attempts to remove and return one piece of candy from the box. The piece of candy to be removed is the first piece of candy with a flavor equal to the parameter `flavor` that is encountered while traversing the candy box in the following order: the last row of the box is traversed from left to right, then the next-to-last row of the box is traversed from left to right, etc., until either a piece of candy with the desired flavor is found or until the entire candy box has been searched.

If the `removeNextByFlavor` method finds a `Candy` object with the desired flavor, the corresponding `box` element is assigned `null`, all other `box` elements are unchanged, and the removed `Candy` object is returned. Otherwise, `box` is unchanged and the method returns `null`.

The following examples show three consecutive calls to the `removeNextByFlavor` method. The traversal of the candy box always begins in the last row and first column of the box.

The following grid shows the contents of `box` before any of the `removeNextByFlavor` method calls.

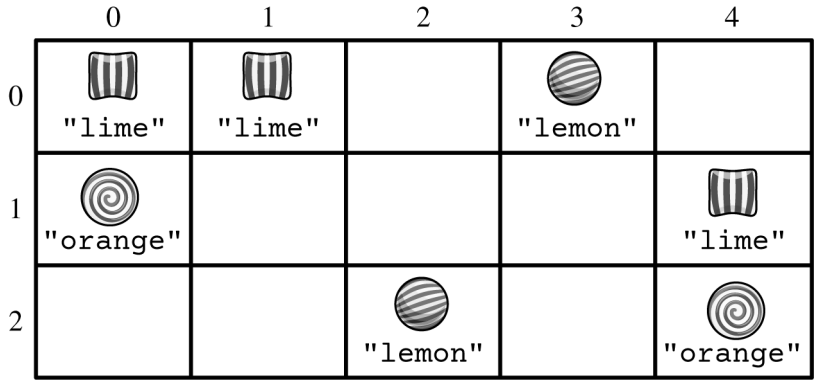
	0	1	2	3	4
0	"lime"	"lime"		"lemon"	
1	"orange"			"lime"	"lime"
2	"cherry"		"lemon"		"orange"

The method call `removeNextByFlavor("cherry")` removes and returns the `Candy` object located in row `2` and column `0`. The following grid shows the updated contents of `box`.

	0	1	2	3	4
0	"lime"	"lime"		"lemon"	
1	"orange"			"lime"	"lime"
2			"lemon"		"orange"

GO ON TO THE NEXT PAGE.

The method call `removeNextByFlavor("lime")` removes and returns the `Candy` object located in row `1` and column `3`. The following grid shows the updated contents of `box`.



The method call `removeNextByFlavor("grape")` returns `null` because no grape-flavored candy is found. The contents of `box` are unchanged.

Complete the `removeNextByFlavor` method.

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

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

Class information for this question

```
public class Candy

public String getFlavor()

public class BoxOfCandy

private Candy[][] box

public boolean moveCandyToFirstRow(int col)
public Candy removeNextByFlavor(String flavor)
```

GO ON TO THE NEXT PAGE.

Question 4: 2D Arrays — 9 points

Canonical solution

(a) — **4 points**

```
public boolean moveCandyToFirstRow(int col)
{
   if (box[0][col] != null)
   {
      return true;
   }

   for (int row = 1; row < box.length; row++)
   {
      if (box[row][col] != null)
      {
         box[0][col] = box[row][col];
         box[row][col] = null;
         return true;
      }
   }

   return false;
}
```

(b) — **5 points**

```
public Candy removeNextByFlavor(String flavor)
{
   for (int row = box.length - 1; row >= 0; row--)
   {
      for (int col = 0; col < box[0].length; col++)
      {
         if (box[row][col] != null &&
             box[row][col].getFlavor().equals(flavor))
         {
            Candy taken = box[row][col];
            box[row][col] = null;
            return taken;
         }
      }
   }
   return null;
}
```

(a) `moveCandyToFirstRow`

	Scoring Criteria	Decision Rules	
1	Accesses all necessary elements of column `col` of `box` (*no bounds errors*)	Responses **can** still earn the point even if they • return early, as long as the loop bounds are appropriate Responses **will not** earn the point if they • fail to access an element of `box` in the loop • access the elements of `box` incorrectly	**1 point**
2	Compares candy box element at row `0` and column `col` to `null`	Responses **can** still earn the point even if they • make the comparison inside the loop or at some incorrect point in the code Responses **will not** earn the point if they • fail to use `!=` or equivalent	**1 point**
3	Identifies and moves appropriate `Candy` object to first row if necessary (*algorithm*)	Responses **can** still earn the point even if they • return early, as long as the identify-and-move are inside a loop and would work if the loop got that far Responses **will not** earn the point if they • fail to replace the moved `Candy` object with `null` • move or swap objects when the first row is already occupied	**1 point**
4	Returns `true` when non-empty square is identified and `false` if non-empty square is not identified in the context of a loop (*algorithm*)	Responses **can** still earn the point even if they • fail to replace the moved `Candy` object with `null` • incorrectly identify a non-empty square Responses **will not** earn the point if they • return early	**1 point**
		Total for part (a)	**4 points**

(b) `removeNextByFlavor`

	Scoring Criteria	Decision Rules	
5	Traverses `box` in specified order (bottom to top, left to right) (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `box` in the loop • access the elements of `box` incorrectly	**1 point**
6	Guards against a method call on a `null` element of the candy box (*in the context of an* `if` *statement*)	Responses **will not** earn the point if they • fail to use `!=` or equivalent	**1 point**
7	Calls `getFlavor` on a `Candy` object	Responses **can** still earn the point even if they • access the element of the candy box incorrectly • call `getFlavor` on the incorrect `Candy` object Responses **will not** earn the point if they • call `getFlavor` on an object of a different type or on the `Candy` class • attempt to create a `Candy` object • include parameters	**1 point**
8	Compares a `Candy` object's flavor with `flavor` parameter	Responses **will not** earn the point if they • fail to use `equals`	**1 point**
9	Replaces first matching `Candy` object with `null` and returns replaced object (*algorithm*)	Responses **can** still earn the point even if they • access elements of the candy box incorrectly • call `getFlavor` incorrectly or on a `null Candy` object • compare a `Candy` object's flavor to the parameter using `==` Responses **will not** earn the point if they • fail to replace the identified object within the 2D array with `null` before returning • fail to return `null` when no matching `Candy` object is found • set multiple elements to `null`	**1 point**
		Total for part (b)	**5 points**

Question-specific penalties
None

Total for question 4 **9 points**