Assignment: `class CandyBox`

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.

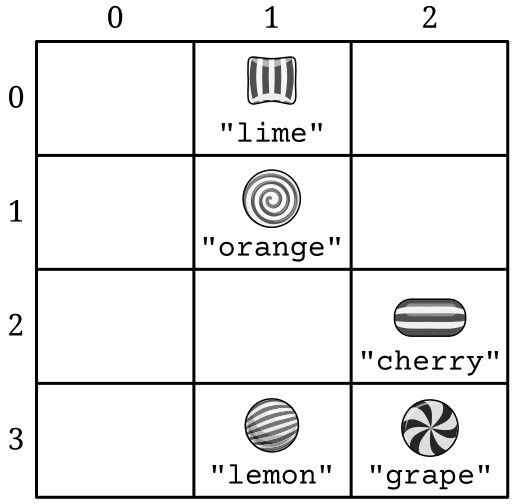
```
public class BoxOfCandy {

    /** box contains at least one row and is initialized in the
      * constructor */
    public 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, 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.

}
```

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`.

Assignment: `class CandyBox`

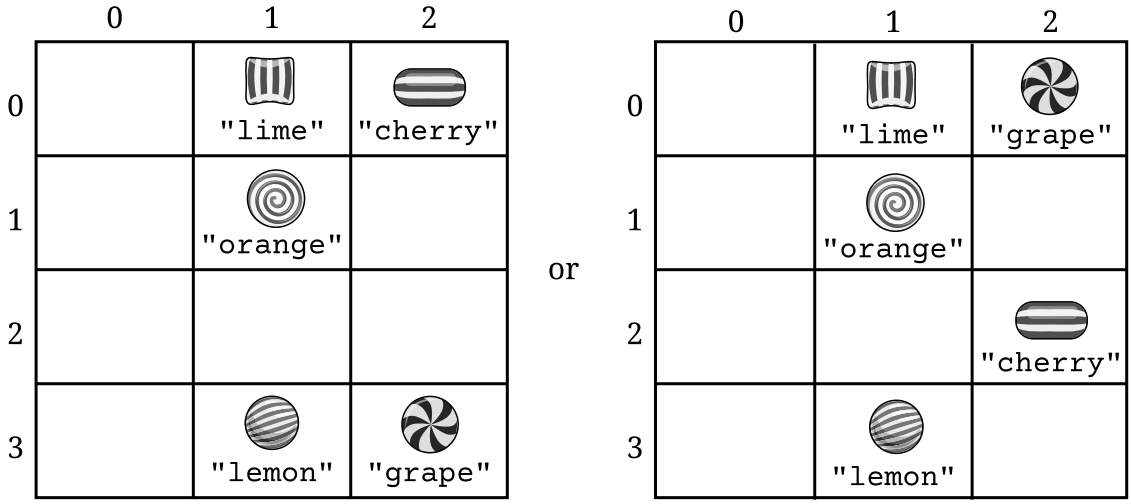
In the following example, the grid represents the contents of `box`. An empty square in the grid is `null` in `box`. A non-empty square in the grid represents a `box` element that contains a `Candy` object. The string in the square of the grid indicates the flavor of the piece of candy.



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.

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.



Assignment: `class CandyBox`

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)
```

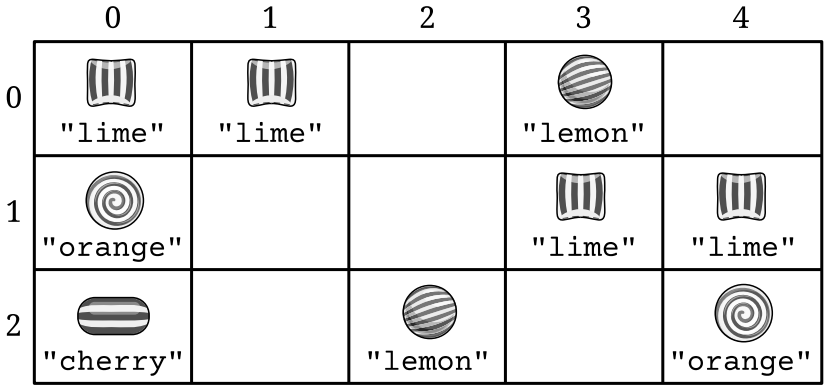
Write the `removeNextByFlavor` method, which attempts to remove and return one piece of candy from the box. The piece of candy to be removed is the first piece of candy with a flavor equal to the parameter `flavor` that is encountered while traversing the candy box in the following order: the last row 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`.

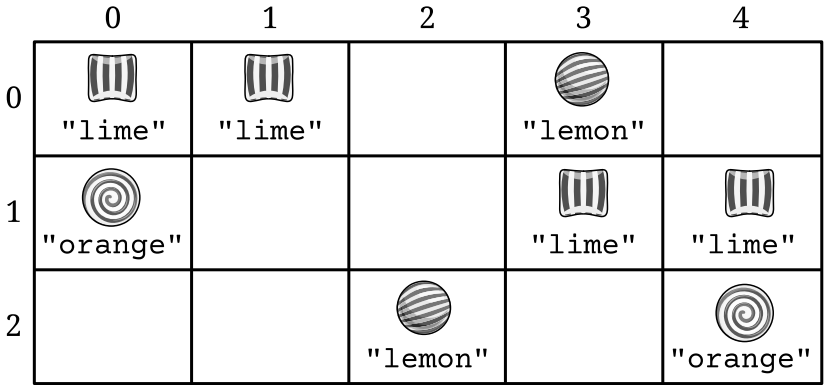
Assignment: `class CandyBox`

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

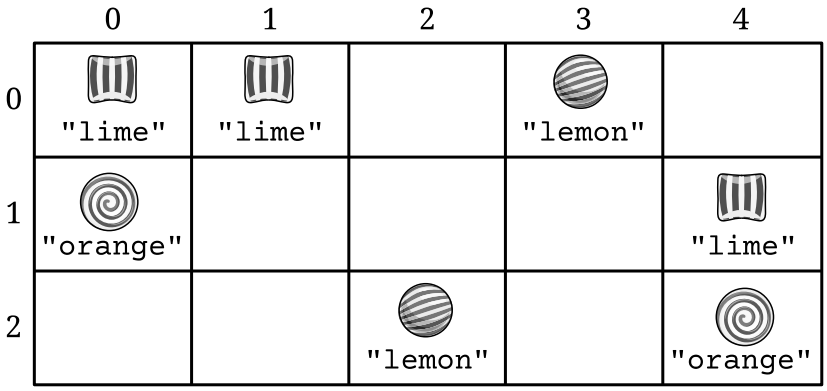
The following grid shows the contents of `box` before any of the `removeNextByFlavor` method calls.



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`.



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



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

Assignment: `class CandyBox`

Complete the `removeNextByFlavor` method.

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