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 )
```

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

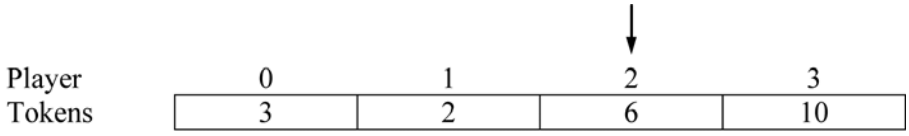
Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered,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

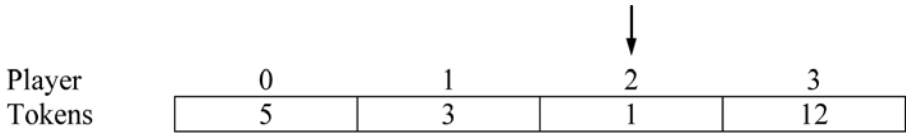
Example
The following represents a game with 4 players. The player at position 2 was chosen to go first.



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.



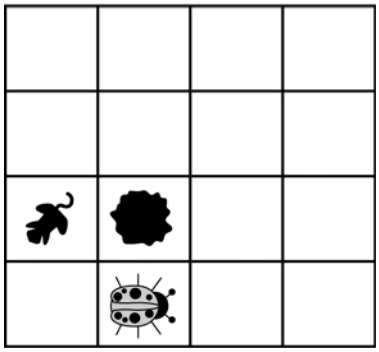
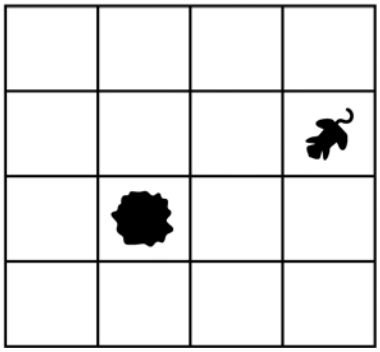
Complete method `getEmptyLocations` below.

```
/** Gets all the locations in grid that do not contain objects.
 *  @param grid a reference to a BoundedGrid object
 *  @return an array list (possibly empty) of empty locations in grid.
 *          The size of the returned list is 0 if there are no empty locations in grid.
 *          Each empty location in grid should appear exactly once in the returned list.
 */
public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
```

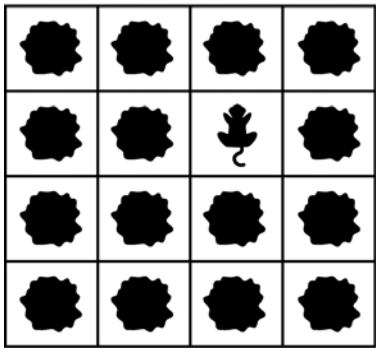
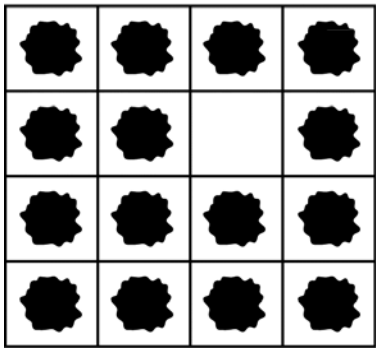
Part (b) begins on page 14.

(b) A `JumpingCritter` acts like a `Critter`, except that it moves by jumping to a randomly selected empty location in its grid. If there are no empty locations, the `JumpingCritter` removes itself from the grid.

The following diagram shows an example of a jumping critter that is able to move to an empty location. Example World #1 is shown below on the left. After the jumping critter at location (2, 0) acts, the world shown below on the right is one possible result.

EXAMPLE WORLD #1	POSSIBLE WORLD AFTER ACT
	
A jumping critter is in location (2, 0).	The jumping critter has eaten the bug that was in location (3, 1) and has moved to location (1, 3).

Example World #2 is shown below on the left. After the jumping critter at location (1, 2) acts, the world shown below on the right is the result.

EXAMPLE WORLD #2	WORLD AFTER ACT
	
A jumping critter is in location (1, 2).	The jumping critter removed itself from the grid because no empty locations were available.

Class information repeated from the beginning of the question

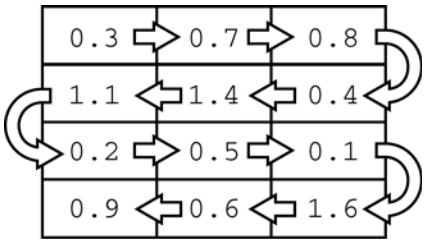
```
public class GridWorldUtilities

public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
```

Assume that the `GridWorldUtilities getEmptyLocations` method works as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality of this method will not receive full credit.

Write the complete `JumpingCritter` class. Do NOT override the `act` method. Remember that your design must not violate the postconditions of the methods of the `Critter` class.

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.
}
```

AP® COMPUTER SCIENCE A
2013 SCORING GUIDELINES

Question 3: JumpingCritter (GridWorld)

Part (a) `getEmptyLocations` **5 points**

Intent: *Create and return* `ArrayList<Location>` *of all empty locations in grid*

+½ Declares and constructs empty `ArrayList<Location>`

+½ Accesses all locations in `grid` *(no bounds errors)*

+2 Identifies empty location in grid in context of loop
- **+1** Creates new location in grid
- **+1** Determines if created location is empty

+1 Includes all and only identified empty locations in constructed arraylist exactly once

+1 Returns the constructed arraylist (*code must have examined grid*)

Part (b) Class: `JumpingCritter` **4 points**

Intent: *Define extension to* `Critter` *class that jumps to randomly selected empty location in its grid*

+½ `class JumpingCritter extends Critter`

+1½ Override `getMoveLocations`
- **+½** `public ArrayList<Location> getMoveLocations()`
- **+½** `GridWorldUtilities.getEmptyLocations(getGrid())`
- **+½** Returns arraylist containing empty locations

+1 Handles `null` location case correctly in `selectMoveLocation`

+1 Handles random location case correctly (must override `getMoveLocations`)

Question-Specific Penalties

-1 (s) Causes inappropriate state change in world (`Grid`, `Actor`, …)

-1 (t) Overrides `act`

AP® COMPUTER SCIENCE A
2013 CANONICAL SOLUTIONS

Question 3: JumpingCritter (GridWorld)

Part (a):

```
public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
{
  ArrayList<Location> locs = new ArrayList<Location>();

  for (int r = 0; r < grid.getNumRows(); r++){
      for (int c = 0; c < grid.getNumCols(); c++){
          Location locToCheck = new Location(r,c);
          if (grid.get(locToCheck) == null){
              locs.add(locToCheck);
          }
      }
  }

  return locs;
}
```

Part (b):

```
public class JumpingCritter extends Critter {

    public ArrayList<Location> getMoveLocations(){
        return GridWorldUtilities.getEmptyLocations(getGrid());
    }

    public Location selectMoveLocation(ArrayList<Location> locs){
        if (locs.size() == 0){
            return null;
        } else {
            Location newLoc = locs.get((int)(Math.random()*locs.size()));
            return newLoc;
        }
    }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.