SNAKE

This problem is based on the game Snake. We are going to simulate that game.

For this problem you will be given a world or 2D array (`String[][]`). The world (the 2 D array) contains one of following Strings:

- A `"F"` representing a food pellet.
- A `" "` (an Empty String) representing a safe location

You have been given the complete `SnakeLocation` class which is used to store the location of all of the snake body parts in the `List<SnakeLocation>`. The `SnakeLocation` class two main methods are `getX()` and `getY()` which return the x and y location. You may add accessor methods, but do not add any methods that modify the `SnakeLocation` object. If you modify the `SnakeLocation` Object, you will need to make a 'Deep' copy of the `ArrayList` passed to constructor in the Snakes class.

The object of the game is for the snake to eat food pellets and grow without hitting the boundary (stay within the world or 2D array) or running into itself.

For this program you will need to implement the constructor and three methods in the `Snake` which keep track of the state of the snake (size of the body, the location of all body parts, and the snake's current direction).

The job of the constructor is to store the world, and the snake body parents. The constructor has two parameters, `String[][] w` and `List<SnakeLocation> sb`. `w` is the current world and `sb` contains the initial snake body.

The world will always:

- Have at least one row (the x-value). `{w.length > 0}`
- Have at least one column (the y-value). `{w[m].length > 0, 0<= m, n < w.length}`
- The world will always be a rectangle `{w[m].length == w[n].length, 0 <= m,n < w.length}`
- `w[m][n].length() == 1`
- `"F ".indexOf(w[m][n]) >= 0`

One additional requirement: **<u>`THOU SHALL NOT modify w`</u>**. That is, you should make your own copy of `w`.

`sb` contains the initial x and y location of all segments of the snakes body.

- the head of the snake is at `SnakeLocation sb.get(0)`
- the segment immediately behind the head is at `SnakeLocation sb.get(1)`
- etc.
- the tail (last segment) of the of the snake is at `SnakeLocation sb.get(sb.size() - 1)`

One additional requirement: **<u>`THOU SHALL NOT modify sb`</u>**. That is, you should make your own copy of `sb`.

Special note: The direction of the snake is a value that must be returned by the next method. You may calculate the direction in the constructor and update the direction as needed, or you may calculate the direction only when asked. It is your choice.

The first method to implement is the `getDirection` method. In this version, snake can only move in one of the four directions: `"U"` (up), `"D"` (down), `"L"` (left), `"R"` (right). The `getDirection` method returns the current direction of the snake (`U`, `D`, `L` or `R`) . The snake is consider to be moving `"U"` (up) if the snake's head is The snake's is on a rwo less than the it second segment. The following code has examples of both move `"L"` (left) and move `"D"` (down).

Sample code follows on next page

SNAKE

The following code shows the results of the `getDirection` method.

The following code	Returns

`String[][] myBoard = { { " ", " ", " ", " "},`
`{ " ", " ", " ", " "},`
`{ "F", " ", " ", " "},`
`{ " ", " ", " ", " "},`
`{ " ", " ", " ", " "} };`
`List<SnakeLocation> mySnake = new ArrayList<SnakeLocation>();`
`mySnake.add(new SnakeLocation(2,2));`
`mySnake.add(new SnakeLocation(2,3));`
`mySnake.add(new SnakeLocation(3,3));`
`mySnake.add(new SnakeLocation(3,2));`
`Snakes sn = new Snakes(myBoard, mySnake);`	

`sn.getDirection();`	`"L"`

The second method to implement is the `move` method. The `move` method moves the snake in the direction indicated by its parameter `m` and returns `true` if the snake was able to move (the snake does NOT run into itself and the snake does not exit the world), and returns `false` if the snake was not able to move (the snake DOES run into itself or the snake moves out of the world). The snake's head should move to its new location according to the following guidelines. If `m` is:

- `"U"` the snake head moves up (from row n to row n-1).
- `"D"` the snake head moves down (from row n to row n+1).
- `"L"` the snake head moves left (from col m to col m-1).
- `"R"` the snake head moves right (from col m to col m+1).
- `" "` (a string containing a single space) the snake head continues to move in its current direction.

As the snaked head moves, each body segment follows the segment directly in front of it by taking on the value of that segment.

If the snake's head moves to a location in the containing `a “ ”` (an empty location), no special action is needed.

If the snake's head moves to a location in the containing `a “F”` (a food pellet), no special action is needed this turn. However, when the snake leaves a location containing `a “F”`, the body of the snake adds one segment. The additional segment is added at the front of the snake. The head moves off the `“F”` and the rest of the body remains in place (The old head is now the second segment of the snake's body. (I have been asked what to do with the "F" in the world. To be honest, we building this problem I was concern only about the Snake and totally forgot about the F. I am not testing the "F", so leave it in the world or delete – it does NOT matter.)

Sample code of this method will follow the description of the third method to be implemented.

The third method to implement is the `getSnakeBody` method. This methods returns a `List<SnakeLocation>` containing all body segments of the snake. The segment at index zero must be the head of the snake, the segment at index 1 must be the segment directly behind the head, and so with the last segment of the snake the tail) at the last index of the `List`.

Sample code of all three methods follows on next page

SNAKE

The following code shows the results of calls to all three methods: `getDirection`, `move` and `getSnakeBody`.

The following code	Returns

`String[][] myBoard = { { " ", " ", " ", " "},`
`{ " ", " ", " ", " "},`
`{ "F", " ", " ", " "},`
`{ " ", " ", " ", " "},`
`{ " ", " ", " ", " "} };`
 	
`List<SnakeLocation> mySnake =`
`new ArrayList<SnakeLocation>();`
`mySnake.add(new SnakeLocation(2,2));`
`mySnake.add(new SnakeLocation(2,3));`
`mySnake.add(new SnakeLocation(3,3));`
`mySnake.add(new SnakeLocation(3,2));`
`Snakes sn = new Snakes(myBoard, mySnake);`	

`sn.getDirection()`	“L”
`List<SnakeLocation> snake = sn.getSnakeBody();`	
`snake.size();`	4
`snake.get(0).getX()`	2
`snake.get(0).getY()`	2
`snake.get(1).getX()`	2
`snake.get(1).getY()`	3
`snake.get(2).getX()`	3
`snake.get(2).getY()`	3
`snake.get(3).getX()`	3
`snake.get(3).getY()`	2
`sn.move("D"));`	true
`sn.getSnakeBody().size();`	4
`sn.getDirection());`	"D"
`sn.getSnakeBody().size();`	4
`sn.getSnakeBody().get(0);`	`new SnakeLocation(3,2)`
`sn.getSnakeBody().get(1);`	`new SnakeLocation(2,2)`
`sn.getSnakeBody().get(2);`	`new SnakeLocation(2,3)`
`sn.getSnakeBody().get(3);`	`new SnakeLocation(3,3)`

More code on the following page

SNAKE

sn.move("L");	true
sn.getSnakeBody().size();	4
sn.getDirection	"L",
sn.getSnakeBody().size();	4
sn.move(" "));	true
sn.getSnakeBody().size	4
sn.getDirection	"L"
sn.move("U");	true
sn.getSnakeBody().size	4
sn.move("R");	true
sn.getSnakeBody().size	5
sn.move("D");	false