Homework1

DDL: Oct.11 23:00,

50% in 24 hours past DDL, 0% after 24 hours past DDL

1. Design a class ***List*** of attributes and methods of linkedlist. Some basic codes are given below. And a test file is also released to help you to test the class.(50%)

```
public class ListNode {
    int val;
    ListNode next;
    //ListNode prev;
    public ListNode(int x) {
        val = x;
        next = null;
        //prev = null;
    }

    @Override
    public String toString() {

        return "["+ val +"]";
    }

}

public class List {
    public ListNode headListNode;
    private int size;
    private int sorted;

    public List() {
        headListNode = null;
        size = 0;
        sorted = 0;
    }

    public List(ListNode node) {
        headListNode = node;
        size = 1;
        sorted = 0;
    }

    public int size() {
```

```
        return size;
    }

    public int sorted() {
        return sorted;//0-unsorted, 1-ascending, -1-descending
    }



    @Override
    public String toString() {
        String str = "[";
        //str += val;
        ListNode pListNode = headListNode;
        while (pListNode.next != null) {
            str += pListNode.val + ", ";
            pListNode = pListNode.next;
        }
        str += pListNode.val + "]";
        return str;
    }

}
```

Methods need to complete:

```
public void sort()
//sort the list ascending. Any sorting algorithm is OK.
//attribute sorted should be changed to 1
public void reverse()
//reverse the order of nodes of list
//attribute sorted should be changed if the list is sorted before
public void addNode(ListNode node)
//add node to tail of list
public void addNodeSorted(ListNode node)
//add node to sorted list and keep list still sorted
//node should add to the position according to the value
public void addNode(int index, ListNode node)
//add node to position of index, which is from 0
public boolean deleteNode(ListNode node)
//delete node, return true if success, false if fail
public boolean deleteNode(int index)
//delete node at position index(from 0), return true if success, false if fail
```

```
public void deleteDuplicates()
//delete duplicated node from unsorted list
public void deleteSortedDuplicates()
// delete duplicated node from sorted list
public void mergeList(List listToMerge)
//merge head of listToMerge to tail of original list
public void mergeSortedList(List listToMerge)
//merge to sorted lists and keep new list still sorted
```

2.Rat in a Maze (20%)

A Maze is a matrix of m*n, with 0 as available blocks and 1 as stones. There may be a path for a rat from position of (startX, startY) to (endX, endY) to eat the cheese. The path may be found by Depth First Searching using stack. The moving order is: right, down, left, up.

```
public class Maze {
    private int[][] mazeMat;
    private int width;
    private int height;
    public Maze(int[][] mazeMatrix, int width, int height) {
        this.mazeMat = mazeMatrix;
        this.width = width;
        this.height = height;
    }
    class Node {
        int x, y;

        public Node(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }

    public void DFSearchPath(int startX, int startY, int endX, int endY) {

    }


    @Override
    public String toString() {
        String ResultString = "";
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
```

```
                ResultString += mazeMat[i][j]+" ";
            }
            ResultString += "\n";
        }
        return ResultString;
    }
}
public class MazeTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int width = 10;
        int height =6;
        int[][] mazeMatrix_1 = {
                { 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                { 0, 1, 1, 1, 0, 0, 0, 0, 0, 0 },
                { 0, 1, 1, 1, 0, 0, 0, 0, 0, 0 },
                { 0, 1, 0, 1, 1, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
            };
            int[][] mazeMatrix_2 = {
                { 0, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                { 0, 0, 1, 1, 0, 0, 1, 0, 0, 0 },
                { 0, 1, 1, 1, 0, 0, 1, 1, 0, 0 },
                { 0, 1, 0, 1, 1, 0, 0, 0, 1, 0 },
                { 0, 0, 0, 0, 1, 1, 1, 1, 1, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

            };

            Maze maze_1 = new Maze(mazeMatrix_1, 10, 6);
            System.out.print(maze_1);

            maze_1.DFSearchPath(0, 0, 4, 4);

            System.out.println("the path is: ");
            System.out.print(maze_1);
            System.out.println();

            Maze maze_2 = new Maze(mazeMatrix_2, 10, 7);
            System.out.print(maze_2);
```

```
            maze_2.DFSearchPath(0, 9, 4, 8);

            System.out.println("the path is: ");
            System.out.print(maze_2);
    }
}
```

Methods need to complete:

```
public void DFSearchPath(int startX, int startY, int endX, int endY)
// show the path from start point to end point with value 2 in matrix.
```

3.Snake Game(30%)
(1) Snake game plays in a m*n grid.
(2) Snake start from (StartX,StartY).
(3) New food generates randomly after the last food has been eaten by the snake. The length get longer after the snake eats the food.
(4) New stone generates randomly every 4*length steps.
(5) The moving order is: right, down, left, up.
(6) Snake dies if it eats stones or itself, or goes out of game area.
The method in Snake.java should be finished. Only 20% if no stones.

```
public Snake(int width, int height)
//generate the game map of matrix
public void StartSnake(int i, int j)
// start snake at position (i,j)
public void ShowSnake()
// print the grid with snake
public void GenerateFood(int x, int y)
// generate food at position(x,y)
public void GenerateFoodRandom()
// generate food at a random position
public void GenerateStone(int x, int y)
// generate a stone at position(x,y)
public void GenerateStoneRandom()
// generate a stone at a random position
public boolean FoodOnSnake(int x, int y)
// if food is on the snake, return true
// food should disappear and generates new food
public void Move(int direction)
// move a step according to direction, 0-right, 1-down, 2-left, 3-right
```

```
import java.util.LinkedList;
```

```java
import java.util.Queue;

public class Snake {

    public int[][] grid = null;
    int width = 20;
    int height = 20;
    Queue<pos> queueSnake = new LinkedList<>();
    pos curPos;
    pos foodPos;

    public Snake() {
        if (grid == null) grid = new int[height][width];
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[i].length; j++) {
                grid[i][j]=1;
            }
        }
        curPos = new pos(1, 1);
        foodPos = new pos(5, 5);
    }



    public class pos {
        public int x;
        public int y;

        public pos(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
}
```

```java
import java.util.Scanner;

public class SnakeTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Snake snake = new Snake(20,10);
```

```
        snake.StartSnake(3, 3);
        snake.GenerateFoodRandom();
        snake.ShowSnake();
        int i=0;
        Scanner sc = new Scanner(System.in);

        while (i<1000){
            System.out.println("the snake is going to move to (1:right, 2:down,
3:left, 4:up):");
            int dir = sc.nextInt();
            if (i%(4*snake.length)==0) snake.GenerateStoneRandom();
            snake.Move(dir);
            snake.ShowSnake();
            i++;
        }

        sc.close();

    }

}
```

Notice:

(1) All the methods should be written exactly the same as the example, including the name, parameters and return type, otherwise it can not run successfully in JUnit tests. Only java codes will be accepted.
(2) All source code in java files should be handed in BlackBoard, including all 3 questions. There should be no package in any class.
(3) The problem 1 and 2 will checked by TAs to run the test file while problem 3 will be checked on lab classes on Oct. 12.