```
import java.util.*;

/**
 * @author ola
 *
 */
public class BlobModel extends AbstractModel {

    protected static final int BLOB_ON = 10000; //'*';
    protected static final int BLOB_OFF = 99999; //' ';
    protected int[][] myGrid;
    protected int[][] myCopy;

    private Random myRandom;

    public BlobModel() {
        myRandom = new Random(1234);
    }


    public void initialize(Scanner s) {
        // nothing to do here, not implemented
    }

    public void initialize(int rows, int cols, int count) {
        myGrid = new int[rows][cols];
        myCopy = new int[rows][cols];
        for (int k = 0; k < rows; k++) {
            Arrays.fill(myGrid[k], BLOB_OFF);
            Arrays.fill(myCopy[k], BLOB_OFF);
        }
        for (int k = 0; k < count; k++) {
            int randRow = myRandom.nextInt(rows);
            int randCol = myRandom.nextInt(cols);
            myCopy[randRow][randCol] = myGrid[randRow][randCol] = BLOB_ON;
        }
        display();
    }

    /**
     * In this model argument is an integer representing minimum blob-size for
     * counting blobs
     */
    public void process(Object o) {
        try {
            int minSize = Integer.parseInt((String) o);
            int rows = myGrid.length;
            int cols = myGrid[0].length;
            for (int k = 0; k < rows; k++) {
                System.arraycopy(myCopy[k], 0, myGrid[k], 0, cols);
            }
            int bcount = 0;
            for (int j = 0; j < rows; j++) {
                for (int k = 0; k < cols; k++) {
                    int size = blobFill(j, k, BLOB_ON, bcount + 1);
                    if (size >= minSize) {
                        bcount++;
                    } else {
                        blobFill(j, k, bcount + 1, BLOB_OFF);
                    }
                }
            }
            display();
            this.messageViews("# of blobs of size " + minSize + " = " + bcount);
        } catch (NumberFormatException e) {
            this.messageViews("illegal argument for counting " + o);
        }
    }

    protected int blobFill(int row, int col, int lookFor, int fillWith) {

        int size = 0;
        if (inRange(row, col)) {
            if (myGrid[row][col] != lookFor) {
                return 0;
            }
            myGrid[row][col] = fillWith; // mark pixel
            size = 1; // count this pixel, then scout for neighbors
            size += blobFill(row - 1, col, lookFor, fillWith)
                    + blobFill(row + 1, col, lookFor, fillWith)
                    + blobFill(row, col - 1, lookFor, fillWith)
                    + blobFill(row, col + 1, lookFor, fillWith);
        }
        return size;
    }

    protected boolean inRange(int row, int col) {
        return 0 <= row && row < myGrid.length && 0 <= col
                && col < myGrid[0].length;
    }

    private void display() {
        this.clear();
        for (int j = 0; j < myGrid.length; j++) {
            StringBuilder sb = new StringBuilder();
            for (int k = 0; k < myGrid[j].length; k++) {
                int val = myGrid[j][k];
                char ch = (char) val;
                if (val != BLOB_ON && val != BLOB_OFF) {
                    ch = (char) ('0' + val);
                    if (val > 9) {
                        ch = (char) ('*' + (val - 9));
                    }
                }
                else if (val == BLOB_ON) {
                    ch = '*';
                }
                else {
                    ch = ' ';
                }
                sb.append(" ");
                sb.append(ch);
            }
            this.notifyViews(sb.toString());
        }
    }

}
```