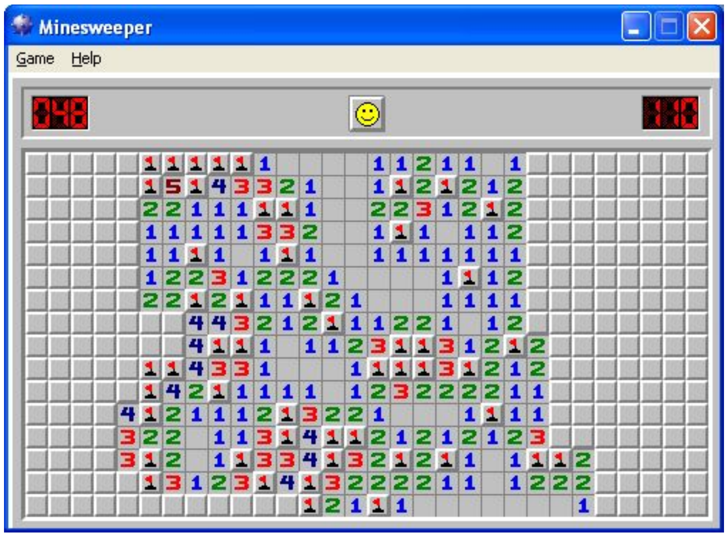
Minesweeper with CLI (Command-Line Interpreter) Interface

In this project you will write a Java program to implement the Minesweeper game. Although the implementation will use a CLI as a substitute for a GUI (graphical user interface), it will be logically quite close to the "real" game.

The game is played by manipulating a rectangular grid of cells. To initialize the game, a subset of the cells is chosen at random to contain mines. Cells that don't contain mines contain an integer that indicates how many covered and hidden mines are adjacent to that cell. Before the grid is shown to the player, all cells are covered to hide their contents (either a mine or an adjacency count). This is the grid that the player sees when the game begins.

To play the game, the player makes a series of moves: either uncover a covered cell by left-clicking it, or mark/unmark a covered cell by right-clicking it. Once a cell is uncovered, its contents are displayed, and it can never be covered again. If the player uncovers a cell that contains a mine, the game is over and the player loses. To win, the player must analyze the adjacency counts of the safely uncovered cells that don't contain mines, and figure out from this which covered cells contain mines and mark them without uncovering them. All other cells, the player must uncover. The player wins when all mined cells are covered and marked, and all other cells are uncovered.



The game requires a mixture of skill and chance. There are times when the user must simply guess that a cell does not contain a mine and uncover it. By comparing the counts of adjacent mines in successfully uncovered cells, the player tries to pinpoint the location of all mines and mark them.

Overview of Implementation
The game is implemented as a grid of cells with properties that can be manipulated by the player from the CLI.

Cell
Each element of the grid is an object of type Cell. A Cell object maintains the following properties:

```
int row, col;      // position in the grid
boolean covered;   // covered is true, uncovered is false
boolean marked;    // marked is true, unmarked is false
boolean mined;     // contains a mine is true, no mine is false
int adjcount;      // number of mines adjacent to this cell (0-8)
```

During a game, the values of most cell properties are fixed before the game begins. How the cell is displayed in the grid depends on the value of these properties. In this program use the following rules:

• covered==true && marked==false:	? (question mark)
• covered==true && marked==true:	X (capital X)
• covered==false && mined==true:	* (asterisk)
• covered==false && mined==false && adjcount==0:	_ (underscore)
• covered==false && mined==false && adjcount>0:	n (integer value from 1 to 8)

Note some restrictions on how the Cell state is manipulated:

- The basic properties are: row, col, mined:
- The adjcount property is used only if mined is false.
- The covered and marked properties interact as follows:
  - To mark a cell, it must be covered. Marking an uncovered cell is an excluded operation.
  - To uncover a cell, it must be unmarked. Uncovering a marked cell is an excluded operation.
  - Once a cell is uncovered, it cannot be recovered.

When initializing a Cell object, the sequence of operations will normally be one of the following

For a Cell containing a mine:

- Instantiate the Cell with a row and col id.
- Set covered to true.
- Set mined to true.
- Set adjcount to -1 (this property is not used for a Cell with mined = true).
- During the game the row, col, mined, and adjcount properties cannot be changed. The cell can be repeatedly marked and unmarked.
- If the Cell is uncovered, the game ends.

For a Cell not containing a mine:

- Instantiate the Cell with a row and col id.
- Set covered to true.
- Set mined to false.
- Set the adjcount value based on the number of mined cells in adjacent positions.
- During the game, the row, col, mined, and adjcount properties cannot be changed. The cell can be repeatedly marked and unmarked.
- If the Cell is uncovered, the adjcount value is displayed.

Phase I: Cell Test

In this version you will write the definition of the Cell class and test it in isolation with a simple CellTest driver class.

The skeleton for the Cell class is as follows:

```
public class Cell {
      private int row, col;
      private boolean covered, marked, mined;
      private int adjcount;
      public Cell(int r, int c) {}
      public void show() {}
      public boolean getMined() {}
      public void setMined(boolean m) {}
      public boolean getCovered() {}
      public void setUncovered() {}
      public boolean getMarked() {}
      public void setMarked(boolean m) {}
      public int getAdjCount() {}
      public void setAdjCount(int c) {}
      public int getRow() {}
      public int getCol() {}
}
```

The full code for the CellTest driver is as follows:

```
import java.util.*;

public class CellTest {
      public static void trace(String msg, Cell c) {
            System.out.print(msg + ":  ");
            c.show();
            System.out.println();
      }

      public static void main(String[] args) {

            Cell c = new Cell(1,3); trace("initial  ", c);
            c.setMined(true);       trace("mined    ", c);
            c.setMarked(true);      trace("marked   ", c);
            c.setMarked(false);     trace("unmarked ", c);
            c.setUncovered();       trace("uncovered", c);

            Cell d = new Cell(2,5); trace("initial  ", d);
            d.setMined(false);      trace("unmined  ", d);
            d.setAdjCount(3);       trace("adj ct 3 ", d);
            d.setMarked(true);      trace("marked   ", d);
            d.setMarked(false);     trace("unmarked ", d);
            d.setUncovered();       trace("uncovered", d);

            Cell e = new Cell(2,5); trace("initial  ", e);
            e.setMined(false);      trace("unmined  ", e);
            e.setAdjCount(0);       trace("adj ct 3 ", e);
            e.setMarked(true);      trace("marked   ", e);
            e.setMarked(false);     trace("unmarked ", e);
            e.setUncovered();       trace("uncovered", e);
      }
}
```

The output from the completed test is:

```
initial  :    ?
mined    :    ?
marked   :    X
unmarked :    ?
uncovered:    *

initial  :    ?
unmined  :    ?
adj ct 3 :    ?
marked   :    X
unmarked :    ?
uncovered:    3

initial  :    ?
unmined  :    ?
adj ct 3 :    ?
marked   :    X
unmarked :    ?
uncovered:    _
```

You should also create your own test cases to show that, for example, a marked cell cannot be uncovered.

Phase 2: Simple Minesweeper

In this phase you will add a MineSweeper class to represent the game board, and a trivial Driver class to play the game.

Class MineSweeper
The MineSweeper class maintains a grid of Cells implemented as a two-dimensional Cell array:

```
int numrows = …;
int numcols = …;
int nummines = …;
Cell[][] cells = new Cell[numrows][numcols];
```

The other instance variables keep track of the dimensions of the grid (rows and columns) and the number of mines.

CLI
The CLI is implemented by a method that accepts inputs from the keyboard that represent the player's next command. The possible commands are:

- show: show the grid by printing it to the display
- q: quit the game
- u x y: uncover the cell at row x and column y
- m x y: mark or unmark the covered cell at row x and column y

The CLI is implemented as a sentinel-controlled loop. The current input is obtained from the keyboard and processed repeatedly until the input represents the "quit" command.

Structure of Program

```
public class Cell { }
public class MineSweeper { }
public class Driver {
      public static void main(String[] args) {
            MineSweeper.play();
      }
}
```

Structure of MineSweeper Class

```
public class MineSweeper {

      private int numrows, numcols, numcells, nummines;
      private Cell[][] cells;

      public MineSweeper() { }
      public int getAdjMineCount(int i, int j) { }
      public int getMarkCount() { }
      public void show() { }
      public void uncoverAll() { }
      public Cell getAdjCell(int r, int c, int direction) { }
      public boolean allNonMinesUncovered() { }
      public boolean allMinesMarked() { }
      public void game() { }
      public static void play() {
            MineSweeper mine = new MineSweeper();
            mine.game();
      }
}
```

Implementation Details

`public MineSweeper() { }`

The constructor creates the grid of cells and initializes their properties as follows. First, each Cell is created for each position in the grid. The properties for each Cell are set like this:

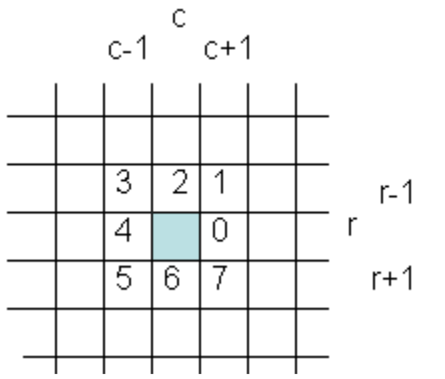
```
row = …;
col = …;
covered=true;
marked=false;
mined=false;
adjcount=0;
```

Next, some subset of the Cells are picked at random to contain mines. The mined property for these Cells is set to true. Once the location of all mined Cells has been chosen, the adjacency counts of all non-mined cells are calculated. This concludes the initialization of the grid. The game is now started by calling the CLI which processes valid commands from the user.

`public Cell getAdjCell(int r, int c, int direction) { }`

This method looks at the cell in the grid at position row=r, col=c. It finds and returns the neighbor of the cell in the direction indicated by the direction input. The direction is a number from 0 to 7, representing the 8 possible directions on the grid from one cell to an adjacent cell. To find the neighbors of the cell at row=r, col=c, you must modify the indexes for row and column by adding and subtracting 1.

There are many ways to number the directions. For example, look at the following figure:



The dark cell in the center is at position row=r, col=c. Its neighboring cells have been arbitrarily numbered from 0 to 7. If you know the position of the dark cell, you can find the position of its eight neighbors as follows:

Direction = 0: adjacent cell to the right of r,c = r,c+1
Direction = 1: adjacent cell to rht upper right of r,c = r-1,c+1
Direction = 2: adjacent cell above r,c = r-1,c
…

Because some cells are on the edge of the grid, a cell may have no neighbor in a particular direction. For example, if row=0, there is no neighbor in the "up" direction. In cases like this, the method should return null. The code should detect this situation by checking if subtracting 1 from a row or column goes below 0. Similarly, code should check when adding 1 if the index goes beyond the actual number of rows and columns in the grid.

`public int getAdjMineCount(int i, int j) { }`

This method is called only for a cell that is not mined. It counts the number of mined cells that are adjacent to the cell at position row=i, col=j. This count will be a number between 0 and 8. It's easy to implement by repeatedly calling the getAdjCell() method for all 8 directions, and counting how many of those neighbors contain mines, then returning that count.

`public int getMarkCount() { }`

This method checks all cells in the grid and counts how many cells are marked. Although the player is supposed to mark only cells that contain mines, this method does not check to see if the marks are correct or not.

`public void show() { }`

This method prints out the current picture of the grid of cells. The show() method of the Cell class will handle the display of each cell based on the values of that cell's properties, as discussed in the description of phase 1.

`public void uncoverAll() { }`

This method is called if the player loses the game to show the contents of all grid cells. It loops over all cells in the board and sets their covered values to false.

`public boolean allMinesMarked() { }`

This method is used to help detect if the player has won the game. It loops over all cells in the grid and checks if each cell with a mine has been marked. It returns true if all mines are marked, false otherwise.

`public boolean allNonMinesUncovered() { }`

This method also helps detect if the player has won the game. It loops over all cells in the grid and checks if each cell that is not a mine has been uncovered. It returns true if all non-mines are uncovered, false otherwise.

`public void game() { }`

The **game()** method implements the CLI (command line interface) with the following algorithm:

```
Create a Scanner to represent the keyboard
Print the game command prompt ">"
Read one line of input
While input is not empty

    Split the line into command and arguments

    If command is show …
    Else if command is uncover …
    Else if command is mark …
    Else if command is quit …
    Else print out "bad command" message

    Print the command prompt ">"
    Read next input

End While
```

CLI Commands

u x y: uncover cell at row x and column y

```
If the cell is marked
    Do nothing
Else if the cell contains a mine
    Uncover All Cells
    Game is declared over, the player loses.
Else
    Uncover the Cell
    If all mined cells are covered and marked, and all non-mined cells are uncovered
        Game is declared over, the player wins.
```

m x y: mark the covered cell at row x and column y

```
If the cell is uncovered
    Do nothing
Else if the cell is already marked
    Unmark it
Else
    Mark it
    If all mined cells are covered and marked, and all non-mined cells are uncovered
        Game is declared over, the player wins
    Else
        Continue
```

show: display the contents of the entire grid of Cells, also print the total number of marked cells and mined cells.

q: quit the game early

Note that there is no command to cover a cell, or to modify the mined or adjcount properties once the game has begun. The mark command acts as a "toggle" so there is no need for an unmark command. To unmark a marked cell, execute the mark command a second time.

Note that as a result of uncovering a cell, the game can be declared a win or a lose. When marking a cell, the game can be declared a win.

Suggestions for Starting Phase 2
To start phase 2, create a simplified version of the MineSweeper class as follows:
- A constructor that initializes a 4 by 4 grid of cells with adjacency count = 0 and covered = true.
- A simplified game() method to support the mark, uncover, show, and quit operations
- A simplified show() method that does not display row and column indexes.

Your Assignment

Step 1: complete the Cell and CellTest classes for Phase 1. Do not submit your work for Phase 1.
Step 2: Copy the Cell class from phase 1 to use in phase 2. Write definitions for MineSweeper and Driver classes.
Step 3: Submit your source files (.java) to the instructor via email.
Subject line: COMP 182 P1
Attachments: all Java source files for phase 2.

Extra Credit
In the extra credit version of the game, an additional operation is performed when uncovering a cell to speed up game play and make the game more similar to the real version. If the uncovered cell is a non-mined cell with adjacency count zero, then all adjacent cells with adjacency count zero (plus a border of cells that surround the zero count cells) should be uncovered together as a group. More detail on how to perform this operation will be provided in a separate posting.

Example Trace for Phase 2

```
> show
Status:  00/04

     0  1  2  3
    ____________
 0|  ?  ?  ?  ? | 0
  |             |
 1|  ?  ?  ?  ? | 1
  |             |
 2|  ?  ?  ?  ? | 2
  |             |
 3|  ?  ?  ?  ? | 3
    ____________
     0  1  2  3
> u 2 0
> show
Status:  00/04

     0  1  2  3
    ____________
 0|  ?  ?  ?  ? | 0
  |             |
 1|  ?  ?  ?  ? | 1
  |             |
 2|  1  ?  ?  ? | 2
  |             |
 3|  ?  ?  ?  ? | 3
    ____________
     0  1  2  3
> u 1 0
> show
Status:  00/04

     0  1  2  3
    ____________
 0|  ?  ?  ?  ? | 0
  |             |
 1|  3  ?  ?  ? | 1
  |             |
 2|  1  ?  ?  ? | 2
  |             |
 3|  ?  ?  ?  ? | 3
    ____________
     0  1  2  3
> u 3 0
> show
Status:  00/04

     0  1  2  3
    ____________
 0|  ?  ?  ?  ? | 0
  |             |
 1|  3  ?  ?  ? | 1
  |             |
 2|  1  ?  ?  ? | 2
  |             |
 3|  _  ?  ?  ? | 3
    ____________
     0  1  2  3
```

```
> u 3 1
> show
Status:  00/04

     0  1  2  3
   ______________
 0|  ?  ?  ?  ?  | 0
  |              |
 1|  3  ?  ?  ?  | 1
  |              |
 2|  1  ?  ?  ?  | 2
  |              |
 3|  _  _  ?  ?  | 3
   ______________
     0  1  2  3
> u 3 2
> show
Status:  00/04

     0  1  2  3
   ______________
 0|  ?  ?  ?  ?  | 0
  |              |
 1|  3  ?  ?  ?  | 1
  |              |
 2|  1  ?  ?  ?  | 2
  |              |
 3|  _  _  1  ?  | 3
   ______________
     0  1  2  3
> u 2 1
> show
Status:  00/04

     0  1  2  3
   ______________
 0|  ?  ?  ?  ?  | 0
  |              |
 1|  3  ?  ?  ?  | 1
  |              |
 2|  1  1  ?  ?  | 2
  |              |
 3|  _  _  1  ?  | 3
   ______________
     0  1  2  3
> u 2 2
> show
Status:  00/04

     0  1  2  3
   ______________
 0|  ?  ?  ?  ?  | 0
  |              |
 1|  3  ?  ?  ?  | 1
  |              |
 2|  1  1  2  ?  | 2
  |              |
 3|  _  _  1  ?  | 3
   ______________
     0  1  2  3
```

```
> m 0 0
> m 0 1
> m 1 1
> show
Status:  03/04

     0  1  2  3
   ______________
 0|  X  X  ?  ? | 0
  |             |
 1|  3  X  ?  ? | 1
  |             |
 2|  1  1  2  ? | 2
  |             |
 3|  _  _  1  ? | 3
   ______________
     0  1  2  3
> u 2 3
> show
Status:  03/04

     0  1  2  3
   ______________
 0|  X  X  ?  ? | 0
  |             |
 1|  3  X  ?  ? | 1
  |             |
 2|  1  1  2  1 | 2
  |             |
 3|  _  _  1  ? | 3
   ______________
     0  1  2  3
> m 3 3
> u 0 2
> u 0 3
> u 1 2
> u 1 3
Status:  04/04

     0  1  2  3
   ______________
 0|  X  X  2  _ | 0
  |             |
 1|  3  X  2  _ | 1
  |             |
 2|  1  1  2  1 | 2
  |             |
 3|  _  _  1  X | 3
   ______________
     0  1  2  3
You WIN!
```