CPSC 170-- Assignment #1

Minesweeper
Due Thursday Jan 31 Noon

Description

Minesweeper is a popular (sometimes addictive) game that uses spatial logic to isolate squares on a grid that hold hidden bombs.

To play:

- The player is initially faced with a rectangular collection of blank squares.
- Upon selecting a square, the game reveals either a bomb or the number of neighboring squares that contain bombs.
- Play continues until a bomb is selected, or until the number of remaining squares equals the number of bombs.

Design Notes

A simple version of Minesweeper can implemented using three major classes:
.

- BombCell.java
    - This class is a special kind of JPanel
    - It has two public final static elements: WIDTH and HEIGHT
    - The class contains the following Instance Data:
        - a JLabel that is either blank, a picture of a bomb or contains a number.
        - ImageIcon to store the bomb image
        - an int to hold the number of nearby bombs.
        - a boolean that tracks if the cell contains a bomb or not
    - The class should support the following methods (with these headers!):
        - BombCell()
        - void setNeighborCount(int n)
        - void setAsBomb()
        - int isBomb()
        - void showContents()
    - The class should support an event-handler to recognize when BombCell has been selected.

- MinesweeperPanel.java
    - This class is a special kind of JPanel
    - Instance data for this class includes:
        - a 2-D array of BombCells
        - Integers to hold the number of rows, columns, and bombs.
    - The constructor for this class
        - Takes three parameters: number of rows, number of columns and number of bombs.
        - Sets its size based on the size and number of the BombCells
        - Initializes the array of BombCells.
        - Randomly assigns the number of bombs.
        - Assigns the neighborCount for each of the BombCells.

- Minesweeper.java
    - Accepts command line parameters
    - Creates a JFrame and sets its contentPane to a MinesweeperPanel.

Implementation Notes

Once you have a clear understanding of the design, you should implement your solution. As usual you should implement this in steps, and test segments of your code for correctness as you are implementing. You should find that a majority of the methods to be implemented are very straightforward and direct. A couple of things to consider

- You can (and should) adjust the font of the JLabel using the setFont method.
- showContents adjusts the jLabel; Changing it from blank (hidden) to reveal either a bomb or a number. Upon completion, call *revalidate()* to make sure that the JLabel is redrawn.
- A bomb is designated by setting an image on the icon (with no text). The image is provided on the course web site.
- When setting the size of your minesweeperPanel, assume there is also a gap of (at least) 5 pixels between the cells.
- When you randomly assign the position of the bomb, you should make sure that you don't accidently assign a bomb to a location that already contains a bomb. When all is said and done, your panel should contain *exactly* the number of bombs specified in the command line.
- The listener associated with your BombCell should implement a MouseListener (we will be talking about that in the next few days (before the due date).
- IMPORTANT: You are **NOT** responsible for writing code to terminate the game (i.e. when a bomb is uncovered or you have exposed all the safe bombCells.)

Suggested Implementation Strategy

1. Create the BombCell class and implement the constructor.
2. Create MineSweeper.java (from cs120 examples)
3. Create MineSweeperPanel.java. – create a small array of BombCells.
4. Make basic adjustments to the size of MineSweeperPanel to get the layout of the board looking right.
5. Start implementing showContents. Create a block of code that displays the contents of all the cells.
6. Add the methods for allowing bombs to appear. Test your code by explicitly setting one of the BombCells to be a bomb.
7. Randomly add the correct number of bombs.
8. For a given BombCell, add the code to count the number of neighboring bombs and assign it to the cell.
9. Remove the code to automatically display the contents of all the cells. Add the listener that triggers a BombCell to transition from hidden to exposed.

Other Requirements: As usual use good programming practices that include appropriate naming of variables, correct indentation, and documentation. You are responsible for producing JavaDoc style comments at the method-level. You should also use internal documentation where necessary to explain what the code is doing.

Turn in: Printed copies of all source code files related to your project. Tar your project directory and send the tar file as an email attachment.

Academic Integrity Reminder: Programming assignments are to be your own work. You may get help on the specifics of the assignment from no one except the instructor. You may not show your program to anyone or look at anyone else's program or share ideas with anyone about how to write the program.