CS 245
MineSweeper

Description:

Minesweeper is a popular (sometimes addictive) game that uses spatial logic to isolate squares on a grid that hold hidden bombs. 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. Players also have the option of "flagging" a square that they have figured out is a bomb. This serves as a visual reminder *not* to select that square. Play continues until a bomb is selected (Player loses) or until the all safe squares are revealed (Player Wins).

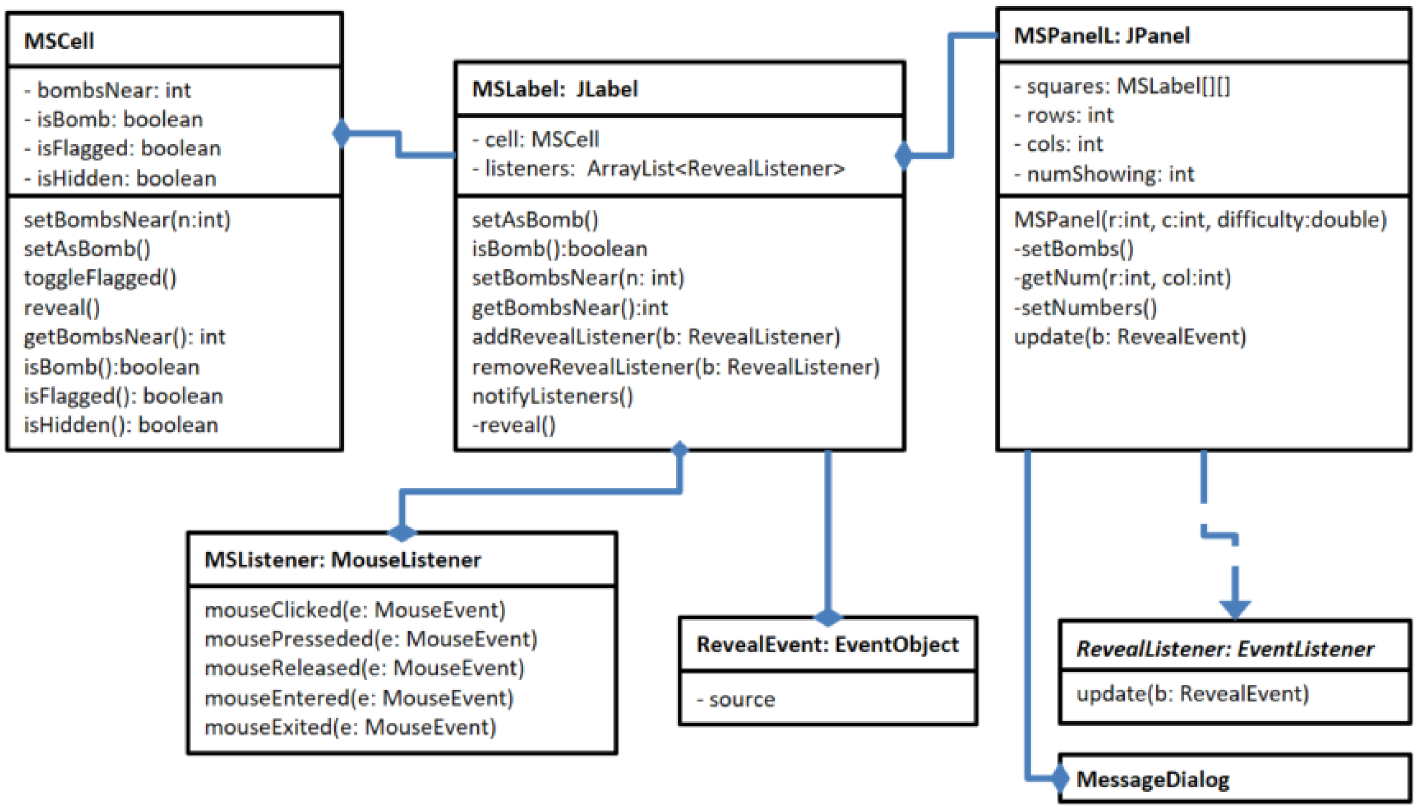
Specification:

You will be responsible for implementing an object-oriented version of MineSweeper that is comprised of the following components:

MSCell models the data for a single Minesweeper Cell. Note that it is independent of the display. Most of the functionality associated with this class are getters or setters. However, you should note that some of the setters are one-way(i.e. setAsBomb and reveal). Likewise toggleFlagged() alternates the isFlagged value between true and false.

MSLabel presents a graphical view of the MSCell. It allows the user to interact with the Cell by displaying the appropriate icons and listening for MouseEvents. This should allow the user to reveal a cell by left-clicking or 'flag' a cell by right-clicking. (Note on Mac, you can use hold the option key while clicking). It is also responsible for Generating RevealEvents when clicking on the label reveals a cell.

MSPanel sets up the game and listens for an ending-condition to be met. When the game is over, it will display a dialog to communicate the outcome and then exit the program.



Roadmap:

At this point I would study the specification and the UML and start to think about how you might code this. Notice that I didn't say, start coding this. Below are some of the kinds of questions that you want to ask (and answer) as a designer ***before*** you write <u>any</u> code. Thinking through these issues will help you develop a plan for how to develop your application.

1. Which class do you start with?

   You could probably be successful with either a top-down or a bottom-up approach.

   - In the top-down approach, you would start with the MSPanel (or actually a JPanel in a Frame). You would start with placeholders that you would gradually replace as you customize components to your needs.
   - Developing from the bottom-up, you would start with the MSCell and try to expand. You start with a concrete data class and gradually add the functionality to manipulate the data.

   In this case, I would probably recommend a bottom-up approach. The base class is very simple and is comprised mostly of getters and setters; there isn't much to do to verify that it works properly and it will give you a good foundation to work with right from the beginning.

2. For MSLabel, why are we extending a JLabel instead of a JButton?

   On first thought, it makes sense for the Cells to be associated with Buttons – after all, we are clicking on components to trigger events – that is practically the definition of a button. Unfortunately, that is all buttons really do – respond to clicks. However, for this application, we need to be able to differentiate between clicking (to reveal) and right-clicking (to flag). So we are going to extend our JLabel with a MouseListener instead (See below for comments about the MouseListener).

3. One way to think of the MSLabel is as an *Adapter* for the MSCell. You already have the data Modeled in the MSCell class, but you need a way to have that communicate with your Panel. Break down the operations of the label into two parts:

   a. **Displaying the state of the MSCell**

      Create a JFrame that has a JPanel that has a bunch of sample MSLabels.
      For each of the MSLabels, manipulate the internal MSCell to display the correct Icons

   b. **Interacting with the user to change the state of the MSCell**.

      Generating a MouseEvent on a MSLabel should change the state of the MSCell and update the icon to reflect that change. Clicking/Pressing/Releasing with the left button should "reveal" what is under a hidden square. Right- Clicking/Pressing/Releasing should set/reset the flag.

4. Once you can get one MSLabel working, you are ready to get a bunch of them going. Now we can worry about the MSPanel. This is the glue that holds the game together.

   a. **Setup the layout**

      Note that the configuration is determined by parameters to the constructor.

   b. **Add the bombs**

      The number of bombs is determined by the size of the board multiplied by a difficulty factor; the difficulty factor is the percentage of squares that should be bombs. For example, if the game is a 10x10 grid, and the difficulty is .2, there are 20 bombs. You should add them randomly, but be careful not to double-up and add a bomb to a square that has already been designated as a bomb.

   c. **Number the remaining squares**

      Each non-bomb square is numbered from 0-8 depending on how many bombs are adjacent to it.

5. The game is starting to look real now, but there isn't anything in place to control the gameplay; If this were implemented (which it isn't because we are still planning), you would be able to click on a bunch of MSLabels and have them respond, but hitting a bomb doesn't do anything, nor is there any way to win. We need a way for the MSLabels to tell the MSPanel that something (good or bad) has happened. To help with this, we will implement the observer pattern (MSLabels are observable, MSPanels are the observers).
   a. **MSLabel needs to be observable**
      Add the necessary code to make the label observable. You must implement
      - instance data
      - addRevealListener()
      - removeRevealListener()
      - notifyListeners()
      - When to call notify Listeners
   b. **Update MSPanells to make it observe the MSLabels**
      - It should now *implement* RevealListener.
        - Start with printing the appropriate message to the console
      - When the MSLabels are created, add the panel as a RevealListener
      - Display the appropriate dialog boxes and end the game.

Mind the Gaps

There are a few language features that you are still learning about:

1. MouseListener
   A MouseListener allows us to respond to five different ways the mouse can interact with a component. Note that you need to "implement" all five methods to satisfy the compiler, but you only need code in the body of the functions that you actually care about. The other function bodies can be left blank.
2. Right-Clicking
   A MouseEvent contains information about the event, like for example, which button was pushed. You can find out by invoking the "getButton" method and comparing it to a built in constant (e.g. MouseEvent.BUTTON1). However this will not work for us; different hardware systems may number the resources differently:
   - What if I have a center scroll-wheel button? Is that Button2 or Button3?
   - What if I am using a Mac that only has one button?

   Mac users know that to right-click on something, they can hold down a key on the keyboard and click. Java sees this key as the "Meta" key. Likewise, when the rightmost mouse button is pushed on a multi-button Mouse, Java sees this as activating the Meta key. So when a MouseEvent e is generated, we can ask it, `e.isMetaDown()`

3. DialogBoxes
   Creating a Dialog Box is accomplished by invoking a static method from JOptionPane. There are two input parameters you need to be concerned with. The first is the "parent Component" – this is the container that the Dialog will be associated with. You can have an independent dialog by passing *null* to this parameter. The second is the message string.
4. Terminating the program: `System.exit(0);` will end the program with at status code of normal.

Optional behaviors (For Bragging Rights)

You are only responsible for implementing the code to this specification. However, there are several features that are common to many implementations of Minesweeper. If you want to consider how to implement these, I encourage you to do so (after everything else is perfect).

1. When the game ends, the entire board is revealed as follows:
   - The explode icon appears on the offending MSLabel.
   - The Bomb icon is used for any unexploded, unFlagged bomb
   - The BadFlag icon is used for any flags that were erroneously placed.

2. Whenever a MSCell is revealed and it have no bombs near it, all adjacent cells are also revealed.

3. When a label that contains a revealed cell is right-clicked, all adjacent cells are revealed.