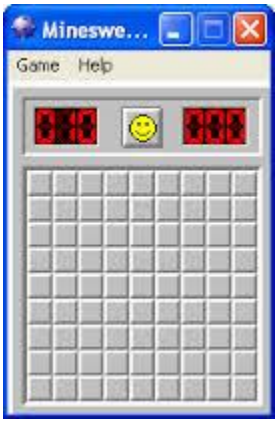
Mine Sweeper

Due: Tuesday, February 23, 2016 at 11:59 pm

For this assignment, you are to implement the game Mine Sweeper using the Java Swing Library. You should be able to find the game on any Windows Machine. You only have to implement the simplest version of the game, which consists of locating 10 mines on a 10 by 10 grid.

This project will be done in teams that will be assigned by the instructor.

An image of the typical Windows game is below. Note that the grid size shown is 9x9 instead of 10x10.



For this program, you will need to click on buttons with both a left and right button. Follow this link for code that shows how this is done.

For the game, you will need to create a playing grid of 10x10 buttons. All 100 of the buttons will have no text displayed on the button. The game begins with the program randomly placing 10 mines "under" 10 of the 100 buttons. The game is over when either the player left clicks on a button containing a mine (the player loses) or when the 90 buttons that do not have mines have been cleared (the player wins).

The description of the program below assumes the program is going to be done using JButtons with text. Using images instead of text earns extra credit (see below) and looks much better.

If the user clicks on a button with the right mouse button, the program is to mark that button as containing a mine. The program is to now change the text of the button from having no text to the character of M (in upper case). This indicates the user believes a mine is hidden under the button. If the user clicks again on this button, the character is changed to a question mark, "?". This indicates the user thinks there may be a mine hidden under the button. If the user clicks on this button a third time, the no text is displayed. Repeated right clicks is to repeat the sequence of characters displayed

from no text to "M" to "?" to no text. While a button is marked with an "M" or a "?", the button cannot be left clicked (see next paragraph).

If the user clicks on the button with the left mouse button, the program must first check what the text on the button is. If the text on the button is "M" or "?", nothing occurs. If the button has no text, the program must check if that button "hides" a mine. If it does, the mine explodes. Then all mines are shown and the game is over. If it does not hide a mine, the button is disabled and the text of the button is to be changed to give the number of mines that are hidden under buttons adjacent to the current button. We will call this "clearing a button". A button can have a maximum of 8 adjacent buttons (those on the edge of the grid will have less). If no adjacent button hides a mine, display the value 0 (See additional comments #1 below for reason on striking out this instruction.) and your program is to automatically clear all adjacent buttons. If one of the buttons that is automatically cleared also has no adjacent hidden mines, this button should also be automatically cleared.

Apart from the 10x10 grid, your program must have another area of the program to display some game information. This information will contain three GUI elements: an output text field that shows the number of mines left to be found, a reset button, and an output text field that shows the amount of elapsed time for the game. These three elements should be separate from the 10x10 grid of buttons (i.e. contained in their own JPanel).

1. The output field that shows the number of mines left should be 10 at the start of the game and decrement by 1 every time a button in the grid is marked with an "M". If the button's text is changed from an "M" to something else, the number of mines left should be incremented. Note simply marking 10 places with "M" does not win the game. The game is won when 90 buttons have been successfully cleared. As soon as the game is won, any buttons that contain mines but are not marked should be automatically marked with an "M" and the number of mines left should be zero.
2. The reset button should allow the user to start a new game at any time.
3. The output field that shows the amount of elapsed time for the game is to display the number of seconds since the user first left clicks on a button in the 10x10 grid. The must be updated as the game is played until either the user wins or loses the game.

Your program must have at least two drop down menus called **Game** and **Help**. In the **Game** drop down menu, you are to have at least three menu items: **Reset**, **Top ten** and **eXit**. Under the Help drop down menu, you are to have at least two menu items: **Help** and **About.** All menu items are to have the mnemonic of the capital letter in the word. The Reset menu items is to perform the same function as the reset button. The eXit menu item should end the program. The Help menu items should display some information about how to play the game (don't go overboard on this, keep it simple!). The About menu item should give some information about the development team (name(s), user-id(s), etc) and course/project information.

The top ten scorers is a list of the top ten fastest users to solve a game. There must also be a way to reset/clear the top ten scorers (perhaps another button on the Game drop down menu). When a user

makes the top ten scorers list, you must prompt for the user's name and record both the user's name and elapsed time for the game. This information must be kept from each time your program is run; therefore, you will need to keep this information in a file (you may assume it is in the java runtime default directory). This file will be need to be read in when the program begins and saved out this information as the program ends. If the file doesn't exist at start time, assume the program is running for the first time and the file does not yet exist.

For 10 points extra credit use images for each button instead of using text to indicate the mine information. You may need to create your own images. However, I am sure you can find such images on the internet.

Your program must be written in good programming style. This includes (but is not limited to):

- meaningful identifier names,
- a file header at the beginning of each source code file,
- a function header at the beginning of the function,
- proper use of blank lines and indentation to aide in the reading of your code,
- explanatory "value-added" in-line comments,
- proper use of classes

The work your group turns in must be 100% from your group. You are not allowed to share code with any other person (inside this class or not) not in your group. You may discuss the project with other persons; however, you may not show any code you write to another person nor may you look at any other person's written code.

Additional Comments and Hints

1. If a button is left clicked and it has no adjacent mines (and is not hiding a mine itself), you will wish to display the text of "0" (zero) instead of having a blank text. By placing a blank text here, you will not be able to distinguish a cleared button versus an uncleared button. This is because both cases have button with no text in them.
2. For the Help menu item, you may make the mnemonic "L" for **heLp** instead of "H" for **Help**.
3. Also, you may not wish to use JButtons in the 10x10 grid. You can add a mouse listener to any swing component you wish. It may work better to have a label instead. You could make an image that looks like a button and initially display that. When a button is cleared, you replace the image with another image that informs the user of the number of adjacent mines. An empty image (that doesn't look like a button) can be displayed when the button is cleared and there are no adjacent mines instead of displaying the value of "0". This idea assumes that you are trying to get the 10 extra points for using images instead of text in the buttons of the 10x10 grid.