PROJECT 6: MINESWEEPER

Milestone: Due March 27, 10:00pm

Final: Due April 1, 10:00pm

TA: Di Xie

WARNING

Your project must compile in order to get a grade. You will get a zero if your program does not compile.

INTRODUCTION

In this project, you will learn how to manipulate Arrays of objects and deal with Exceptions. You will also gain more experience with graphical user interface (GUI) objects.

BOOK CHAPTERS

You should read chapters 8 and 10 before beginning this project. Dealing with Arrays involves many subtle issues and reading the book before you begin will help save debugging time. In addition, you might want to reference Chapter 14 of the book for help with Event-Driven Programming and GUI aspects.

OVERVIEW

Minesweeper is a single-player computer game. The object of the game is to clear an abstract minefield without detonating a mine. The game has been written for many system platforms in use today- including the Minesweeper for the Windows platform, which has come bundled with versions of the operating system from 3.1 and on. In Solaris Systems, you can start this game by typing /usr/sfw/bin/gnomine in the terminal, or you can find it in the Launch menu if you are using Sun Java Desktop.

In the game, there are a number of mines in the field. Your job is to try to find the locations of all the mines without getting blown up:

- You can uncover a square to look for clues by left clicking it. If you uncover a mine, you lose the game.
- If a number appears on a square when you uncover it, it indicates how many total

mines are in the eight squares that surround the numbered one. You can use this number to help deduce whether a square is safe to uncover.
- To mark a square you suspect contains a mine, right-click it. This will add a flag to the square.

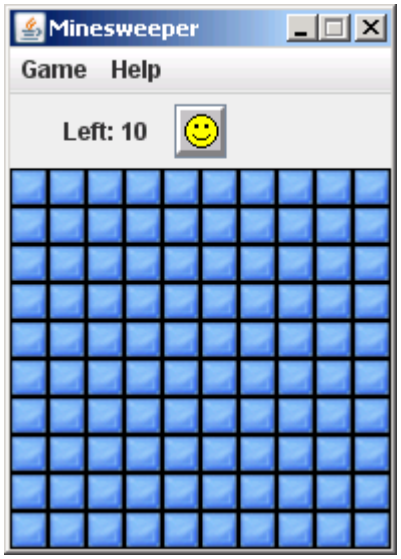
You can find more information about Minesweeper here.

You have learnt a lot about GUI, exceptions and arrays, it's time to show your programming ability and make your own version of minesweeper! Since this is your first game written by yourself, you are only asked to write a simple version. Below is the specification.

MINESWEEPER SPECIFICATION DETAIL

Initially your program should present a window with the following:

- A title that says Minesweeper
- A menu that includes two options: Game and Help
- A label that says "Left: " and a counter indicating the number of squares yet to be flagged
- A button with a smiley face indicating the status of the game
- A 10 X 10 grid of blank covered squares



MENU

There are 4 menu items under the Game menu:

- **Beginner** should restart the game in the beginner mode, in which there are exactly 10 mines.
- **Intermediate** should restart the game in the intermediate mode, in which there are exactly 20 mines.
- **Advanced** should restart the game in the advanced mode, in which there are exactly 30 mines.

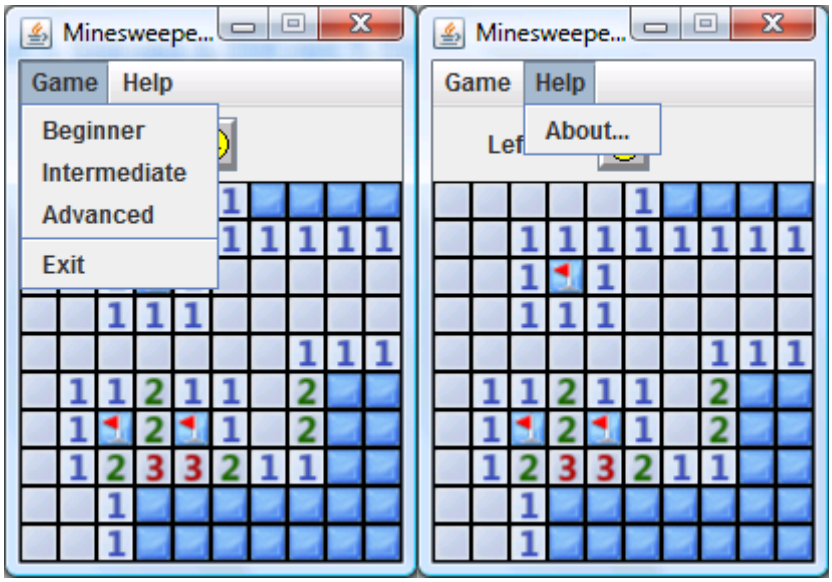
- **Exit** should Exit the game. the game.

> Note: There is a separator between **Advanced** and **Exit**. And when a user chooses ANY level of difficulty (including the current level) by clicking the menu item, the game restarts. The default level is **Beginner** when the program starts.

There is only 1 menu item under the Help menu:

- **About** should Show the author information of the program in a pop-up window. And you can add other information if you want.

Below are 2 sample screens showing the drop down menu for Game and Help:



FACE BUTTON

This button plays an important role in the game:

1) You can restart the game in the current level of difficulty by clicking the button any time.
2) The face shows the state of the game:

-  means the game is in progress
-  indicates that the user won the game
-  indicates that the user lost the game.

3) When the game is in progress, if a user is pressing any button of the mouse in the window, the face icon becomes  (in anticipation of ... disaster? reprieve?) And when the button is released, the icon becomes  again.

COUNTER

There is a counter to the left of the face button, which indicates the number of squares yet to be flagged. It is the number of mines minus the number of flagged squares, so when you flag a square, the number decreases by 1; when you unflag a square, the number increases by 1.

> This counter does NOT indicate the number of mines that have not been flagged, since some of the flags may be misplaced. The counter may also become a negative number when a user flags more squares than the number of mines.

PLAYING FIELD

The play field is 10 rows by 10 columns. At first, all the squares in the field are covered. As long as the game isn't over, a user can click in the field. Below is the detailed description of the behaviors of clicking the left mouse button and clicking the right mouse button.

Left click	Action
Unflagged and covered square	The square becomes uncovered. Also ... • If the square contains a mine, the mine explodes and the game is over. • If there is no mine in the square, but there are mines in the eight squares that surround the square, the square displays a digit indicating the total number of mines in the surrounding squares. • If there is no mine in the square, and there is no mine in any of the surrounding squares, the adjacent squares of each of the 8 adjacent squares of the current square will be uncovered automatically. If an adjacent square is still empty, the adjacent squares of that will also be uncovered. This process continues until all the adjacent empty squares are uncovered. (This process is a little complex for you, so you are provided a method **`expandSquare(int row, int col)`** which you can call. When you feed the position of an empty square to the method, it will expand all the squares according to the rule stated. For more detail, please refer to the comments in the `GameMaster.java` file.
Flagged square	A flagged square cannot be uncovered. The user will be presented with the following message: "You cannot overturn a flagged square!"

Uncovered square	An uncovered square cannot be uncovered again. The user will be presented with the following message: "You cannot overturn an uncovered square!"

Right click	Action
Covered square	• If the square is unflagged, flag the square. The counter decreases by 1. • If the square is already flagged, unflag the square. The counter increases by 1.
Uncovered square	An uncovered square cannot be flagged. The user will be presented with the following message: "You cannot flag an uncovered square!"

WIN OR LOSE?

If users uncover a square containing a mine, they lose the game. And they should be notified by the message "You lose!"

If users finally UNCOVER ALL the squares without mines, they win the game. It is NOT necessary to flag all the mines to win the game, which means they can even win the game without flagging any squares. Conversely if they have only flagged all the mines but haven't uncovered all the squares without mines, they don't win the game. That is, the user MUST UNCOVER ALL THE SAFE SQUARES TO WIN! The program should pop up a window displaying "You win!" when users win the game.

When the game is over, the playing field should not respond to users' action any more. If users click (either left-click or right-click) in the field, they should get either of these messages:

- "The game is over. You have won! Click the face button to play a new game."
- "The game is over. You have lost! Click the face button to play a new game."

In addition, if a user lost, we should tell him which mines he didn't find and which flags he marked at a wrong square:

- Show a  in the square indicating a mine they didn't find.
- Show a  in the square indicating a wrongly flagged mine.
- Show a  in the square indicating a successfully flagged mine.
- Show a  in the square indicating the explosive mine triggered by left-clicking.
- For other squares, keep the same icons as the game in progress: Show digits for uncovered squares, and show intact icons for covered squares.

Following is an example when users lose a game:

IMPLEMENTATION DETAILS

You must implement 6 classes:

- **`Project6.java`**
- **`MainFrame.java`**
- **`GameMaster.java`**
- **`GamePanel.java`**
- **`Square.java`**
- **`InvalidClickException.java`**

In addition, you are provided a class `State(State.java)`, which contains several constants used in other classes.

The Java Doc for all the classes have been provided. You can read them to understand the relationship between the classes. However, these documents just provide you with one possible solution to the project. You can implement your classes differently from those methods specified in the documents.

You can download the skeleton code, all the necessary icons and the GameMaster.class here

THE PROJECT6 CLASS

The Project6 class should have the **`main()`** method, which starts the game.

THE MAINFRAME CLASS

The class creates the main frame (window) of the game.

- The title should be set to "Minesweeper".

- It is positioned at screen coordinate (200, 150).
- It should create all the menus and menu items mentioned in the specification.
- It contains the `GamePanel`, the counter (of class `JLabel`) and the face button (of class `JButton`).
- It listens to the mouse event for the face button and the action event for menu items (`ActionListener`).
- You can specify a proper size of the window to make it good-looking. However, you can call `pack()` instead, after you have added all the components to the frame, in order to make the window fit the inner components better.

THE GAMEMASTER CLASS

This class implements most of the game logic, and maintains a lot of the data used in the GUI, some of which are stored in arrays. You are provided some skeleton code for this class. Please read the code and comments to get a better idea of the class.

Two arrays are needed to represent a square: One array, `state`, indicates the state of the square, e.g. it can be `Covered`, `Uncovered`, or `Flagged` (They are defined in class `State`). Another one, `digit`, indicates how many mines are in the eight squares surrounding it, but if the square itself contains a mine, it contains a special number, -1. (Actually, this special number has been defined as `MINE` in the class `State`).

You are NOT required to finish this class for the MILESTONE. And you are provided a functional class without the source code in the form of compiled class file (GameMaster.class), so you can directly use the methods in the class. Please refer to the comments in the skeleton code for the function of each method.

> Note: The provided class only has a fixed layout of mines. Your class should be able to randomize the layout in the FINAL submission.

> Note: Please do NOT compile your GameMaster.java to avoid replacing the provided GameMaster.class, when you are programming for the MILESTONE.

For the complete project, you must finish this class and submit your code along with the other class files. In the `uncoverSquare()` and the `flagSquare()` methods, you should throw an `InvalidClickException` if the square cannot perform the action (as in the specification of Playing Field). The exception should include the state of the square and the state of the game.

> Note: Please make sure that you are using your own GameMaster class, when you tests your FINAL project.

THE GAMEPANEL CLASS

This is the graphical representation of the playing field, which is made up of 10 X 10 squares. Refer to Ch14TicTacToePanel.java, you will find it helpful when you are writing this class. In this class, you should:

- Create all the squares in the constructor, and for each square, you should tell the `GameMaster` its position by calling the `setSquarePos()` method of `GameMaster`. Then you can get a square by calling the `getSquareAt(row, column)` method later. In addition, you have to register the mouse event listener for each square.
- Write a method to refresh the playing field and make every square display an icon consistent with its state.
- Deal with mouse event: Implement `mousePressed`, `mouseReleased` method, because when you press a mouse button down, the face button should show  , while you release the button, it becomes  again.
- When a left click is detected, uncover that square by calling `uncoverSquare()` in `GameMaster`. When a right click is detected, flag that square by calling `flagSquare()` in `GameMaster`. Hint: You can implement it in `mouseClicked`. (But you may find the respond a little dull when you click frequently; you can avoid that by implementing it in `mouseReleased`. The reason is when you act swiftly, you are actually dragging the mouse instead of clicking, which cannot be detected by `mouseClicked`.)
- `uncoverSquare()` and `flagSquare()` will throw an `InvalidClickException`, if the square cannot be uncovered or flagged. Thus, you should catch the exception and pop up a message as specified to warn the user.
- If users finally win, they should be presented with a message "You win!"; if users lose by uncovering a mine, they should be presented with a message "You lose!".

THE SQUARE CLASS

Each object of this class corresponds to a square in the playing field. It plays a similar role as the Ch14TicTacToeCell class in the TicTacToe Game, so you should study the Ch14TicTacToeCell.java file and make sure you understand it.

- In the constructor, it should store the position in the playing field, so that the `GamePanel` can get the position when it processes mouse event.
- It provides methods that can return the position of the square, as mentioned above.
- It provides methods to set icons according to the state (`Covered`, `Uncovered`, `Flagged`, or show the mine icons  when the game is over).

THE INVALIDCLICKEXCEPTION CLASS

Because we want to utilize the exception to get extra information about what scenarios led to the exception, we need to define our own exception. The exception you define in this class will be used in the `GameMaster` class. The constructor is given in the skeleton

code; you should complete it, and add methods based on how you handle this exception in the `GamePanel` class.

THE STATE CLASS

This class contains several constants that can be used in the project.

- `WIN`, `LOSE` and `INPROCESS` denotes the state of the game.
- `SQUARE_COVERED`, `SQUARE_UNCOVERED` and `SQUARE_FLAGGED` are used to represent the state of the square.
- `MINE`: If a variable digit has the value `MINE`, it indicates that the square contains a mine.

CORRECTIONS/CLARIFICATIONS

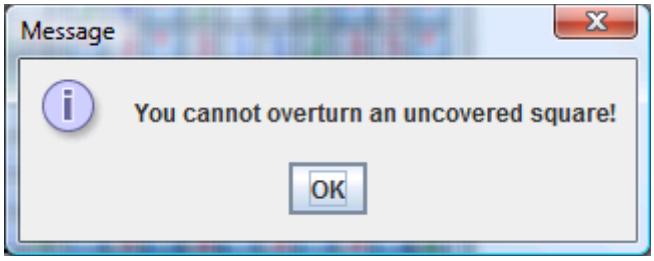
On class newsgroup. Important ones will be posted here too.

MILESTONE (30% OF PROJECT GRADE)

You must implement the GUI part of the game for the Milestone portion of the project. Since you haven't learnt Arrays yet, you are not required to implement the major logic residing in the GameMaster class. Instead, you are provided a compiled GameMaster.class to help you check your codes for the GUI. Make sure your program can achieve all the functions in the Overview part.

Because the layout of the mines for each level is fixed for the milestone (hard-coded in GameMaster.class), you should be able to come to the following phase for the beginner, intermediate and advanced level respectively, if you win the game.

All the messages should be in a pop up window like the following:



MILESTONE SUBMISSION

The milestone files must be turned in electronically by the deadline specified at the top of this page. You should turn in all the following Java source files:

- Project6.java
- MainFrame.java
- GamePanel.java
- Square.java
- InvalidClickException.java

We should be able to execute your project by compiling the above Java files you submitted and running the Project6.class file. (We will use the same GameMaster.class when we test your project.)

Use the following command from the directory of your project to turn in your milestone:

```
turnin -v -c cs180secXXXX -p project6m *.java
```

where XXXX is the division and section number for your lab such as 0101, 0201, 0301, etc.

FINAL (70% OF PROJECT GRADE)

Your final project should implement GameMaster.java. And now you can play the game using your own code. Make sure you can generate a correct layout of the mines, which contains a number of mines according to the difficulty level and the digit in each square should be consistent with the number of mines surrounding the square.

PROJECT SUBMISSION

The project must be turned in electronically by the deadlines at the top of this page. You should turn in all of your Java source files (including GameMaster.java). We should be able to execute your project by compiling all the Java files you submitted and running the Project6.class file. Project turnin is similar to lab turnin. Late submissions and fixing incorrect submissions (e.g. submitting the wrong file) will not be accepted. Remember that each submission overwrites all previous submissions. Prior submissions cannot be retrieved.

Use the following command from the directory of your project to turn in your project:

```
turnin -v -c cs180secXXXX -p project6 *.java
```

where XXXX is the division and section number for your lab such as 0101, 0201, 0301, etc. *Note* that these numbers correspond to the bold numbers associated with your lab on the Lab Divisions page.

If you are having trouble submitting your files, talk to your TA for help.