The SOS Game

There are lots of ways to build working programs, some better than others especially when it comes to managing change. Even though its core functionality may remain the same, how users see and interact with an application can vary dramatically with new technologies, and with market and fashion trends. Separating a program's logic from its user-interface thus makes sense, allowing such changes to be isolated. This is the central idea behind the Model-View-Controller (MVC) design pattern, which not only isolates the model from the view, but ultimately allows for multiple views and even multiple controllers.

As a simple demonstration of this idea you are asked you to construct the SOS game, details of which can be found in this Wikipedia article. You are given the completed SOS Java class -the model- and asked to create a simple GUI -the view & controller- for it. The completed game will look something like Figure 1.

As always, begin by creating a new project in your favourite IDE. The first thing is to make sure you can instantiate and use the *SOS* class. Download the SOS.class file from Moodle and save it somewhere on your IDE's classpath. *Hint:* since it's full name is "cs101.sosgame.SOS", it should probably be in a folder called "sosgame", which is inside one called "cs101". In a main method, create a new instance of the *SOS* class and make sure your program compiles and runs properly. Refer to the UML diagram in Figure 2 for details of the *SOS* class constructors and methods. You might like to make a simple console-based (text-only menu-driven UI) as one example of an application using this model. Of course, nowadays most people expect a "fancy" graphical-user-interface (GUI), so go ahead and give them one, with MVC it's easy!

Building the game's GUI:

Constructing GUIs is usually done in two stages: (1) creating the UI, and (2) add the interactions so that everything works. As far as possible, you should also be building reusable UI components so you can add any number of them to JPanels, JFrames, or JApplets.

(1) Creating the UI...

Create a new Java class called *SOSCanvas*, that extends *JPanel*. This class will simply display the SOS game as a grid. While you could use a collection of JButtons in a GridLayout --as you did in Lab04 part a-- this time we want you to draw the grid yourself. To do this you will need to override the *paintComponent* method. You will also need a reference to the SOS game, so have the constructor take a reference to an instance of the *SOS* class and save it into a suitably named property. Test the class by having your main method create an instance of your *SOSCanvas* class (passing it an instance of the *SOS* class), and add it to a *JFrame*.

Note: you should be able to call the SOS play method a few times before making the frame visible, so you can see the letters are correctly positioned in the grid.

Next, create a class called *SOSGUIPanel*, that again extends *JPanel*. This class will include an instance of the *SOSCanvas* class, as well as the other UI components necessary to show the player names and scores, and the radio buttons for selecting the letter to play. You should pass an instance of the *SOS* class and two Strings containing the players names, to its constructor. Test this class by adding an instance of it to the *JFrame* in your main method. Once you have the GUI looking like that of Figure 1, it's time to wire everything up so it actually works!

(2) Adding the interactions...

Having got the basic UI done, you can start "wiring it up" using Java's event handling mechanism.

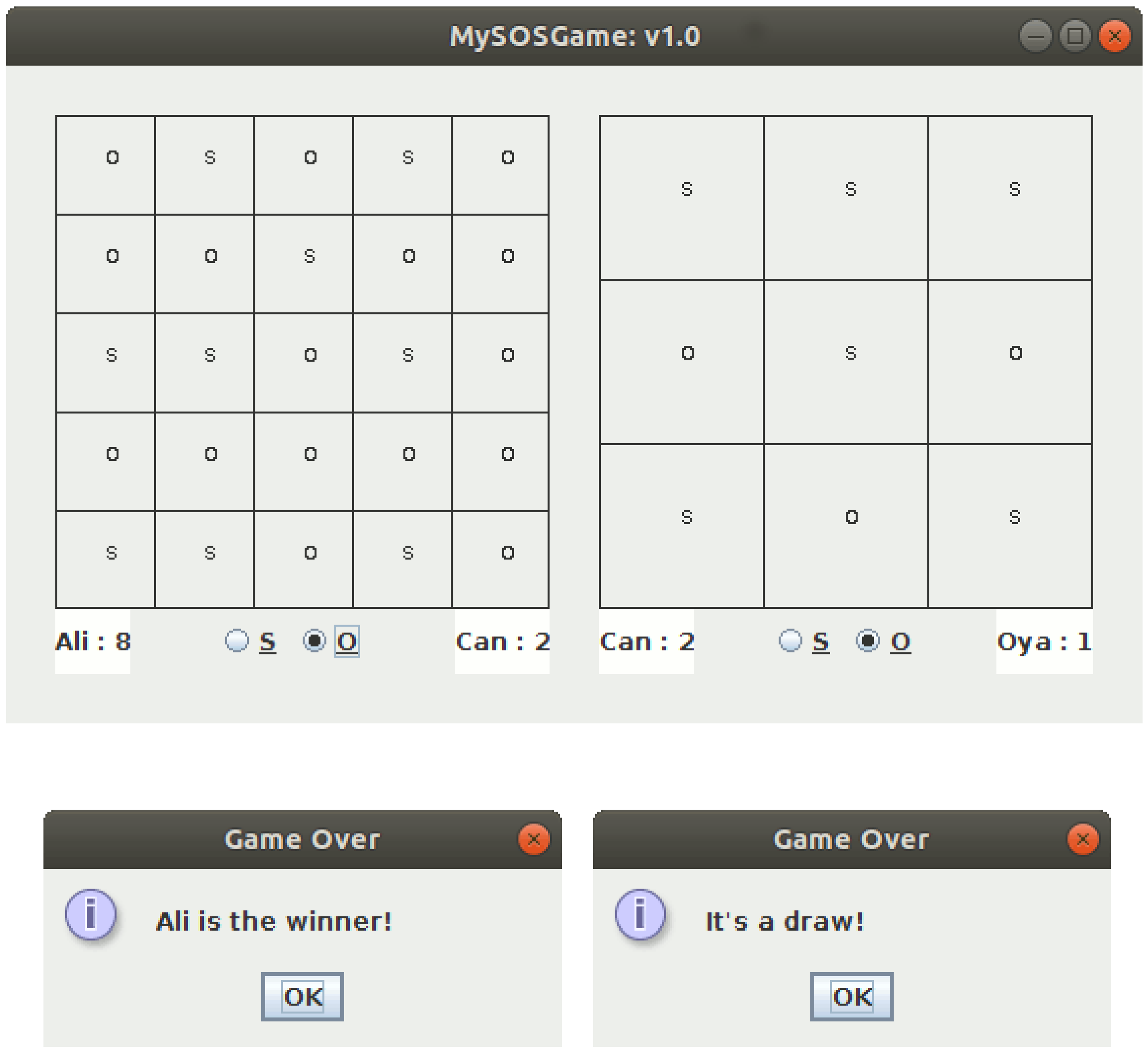
Actually, you only need to be able to detect mouse events on the *SOSCanvas*. When the user clicks the mouse on a particular cell on the grid, you have to determine its row and column number, then, using this information and the current state of the radio buttons (that say which letter the user wants to play), call the SOS play method and update the (grid and the score) displays accordingly --changing the background colour of the player name/score labels so as to indicate whose turn it is. When the game is over, (use a *JOptionPane* to) pop-up a message saying who won. *Hint:* there are a lot of ways to code this, but the simplest solution is probably to write the mouse listener as an inner class of the *SOSGUIPanel* class and add an instance of it as a listener to the *SOSCanvas* instance in the *SOSGUIPanel* constructor.

Congratulations

You should now have a completely functional SOS game. Ok, it may not be very pretty, but it does work and, more importantly, you should have learnt a bit about creating such programs and demonstrated the advantages of the MVC pattern. If you have time you might try making it even better. For example: try changing the text size and font used to display the letters in each cell as well as the player names and scores; add a border to the radio button panel; allow players to enter their names and to choose the grid size they will use (there are a lot of UI options for achieving this --menus, dialogs, panels, with textfields, choiceboxes, lists, etc.-- the more of them you try the more you will learn). You might also try extending the game itself, perhaps by adding a time limit for players to move or by placing a few letters on the grid at random before play begins. Players should be able to start a new game or close the program whenever they wish, but it might be nice to remind them that the current game will be lost (assuming it hasn't yet finished). Alternatively, you could automatically save the game state and reload it next time the program is opened. Even more sophisticated, draw coloured lines showing the places where each player scored points. *Note:* some of these may require "modifying" the SOS class!

Use your imagination and show us what you can do!

Figure 1. Two instances of the basic SOS game in one frame and two example "Game Over" messages.



O	S	O	S	O
O	O	S	O	O
S	S	O	S	O
O	O	O	O	O
S	S	O	S	O

S	S	S
O	S	O
S	O	S

Ali is the winner!

It's a draw!

Figure 2. UML diagram for SOS class.

<u>The SOS class:</u>

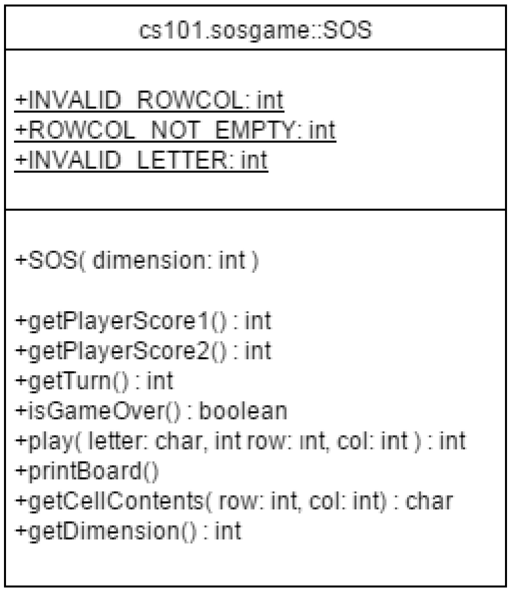
In package: cs101.sosgame

getTurn() : int ~ returns player number, either 1 or 2

getDimension() : int ~ returns grid size

printBoard() ~ shows current board state on Java console

play(letter, row, col) : int ~



attempts to place the given letter --lower case 's' or 'o'-- at row, col on the grid

- returns: points [0..8] if successful,
  else negative value indicating error [INVALID_ROWCOL, ROWCOL_NOT_EMPTY, INVALID_LETTER]
