CSC2018
GUIs, Arrays and Data States

Objectives:

- To understand how to use arrays of components, including their event handling.
- To become familiar with more complex programs involving GUI components.

In QoL under ***Resources...Advisories...Week5*** you will find a zip folder ***Week 5 Software.zip***. Save it to your computer and unpack it (right click...***Extract All***...). In the folder ***Week 5 Software*** you will see four zipfiles. These are the project archives that you will use for this week's exercises. Import the projects into Eclipse in the same way as you did with the software for Week 3. The java files mentioned below contain the main method of each project.

Now try the exercises.

1. An array of buttons (LabTesta1.java)

Program **LabTesta1.java** contains two versions of the same frame – one using five individually named JButton objects (myFrame1), the other using an array of five JButtons (myFrame2). The event handling in both cases prints a string to the *Console* window. Execute the program, and try clicking all the buttons. Note what happens, and what output you see displayed.

Now look at the program – first *LabTesta1.java* (the easy bit). Then look at the non-array implementation (*myFrame1.java*). The five buttons are declared as:

```
JButton b0, b1, b2, b3, b4;
```

Make sure you understand exactly how the event handling operates in this version.

Now look at the code in *myFrame2.java*. Note that the button array is declared here as:

```
JButton [ ] b;  // for an array of buttons - not yet created
```

Buttons are actually created one by one in a loop:

```
for (int i=0; i<5;i++)
{
      b[i] = new JButton("Name "+i);
      b[i].addActionListener(H);
      c.add(b[i]);
} // for
```

Note how the labels for these buttons are generated inside the loop. Compare this with the labels which appear when the program is executed.

Now examine the event handling. Trace through what happens when the second (middle) button is clicked. What code is activated? How is this button event processed?

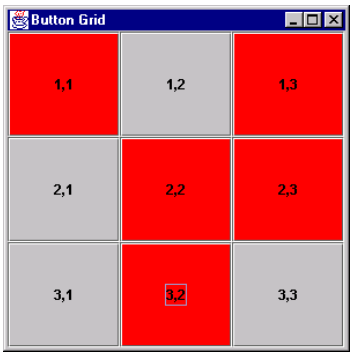
Exercise: Extend the array version of the program so that, if you select an even-numbered button (0, 2 or 4), it also outputs the string "Bingo!".

2. Multiple mini-panels (LabTesta2.java)

Compile and execute the program *LabTesta2.java* first, to see what it displays on the screen. Then open the two java files, *LabTesta2.java* and *myFrame.java*. The program creates an array of buttons, text fields and panels. Each panel object (which doesn't have a name) is composed of a button b[i] and a text field t[i]. A little function (miniPanel) builds a new panel with its two components. Note that the main program says how many groups it wants, by passing this in to myFrame as a parameter.

Exercise: Change the parameter in the 'main' method to get a different number of panels. Make sure you understand the purpose of every line in the program!

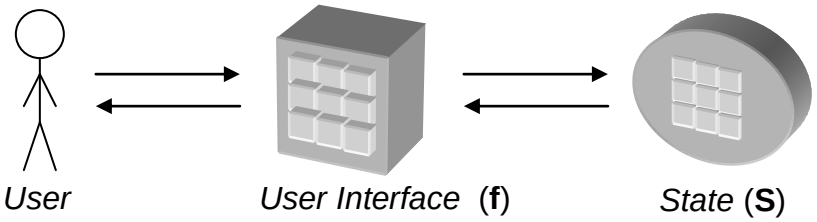
3. Introducing data behind the interface (LabTesta3.java)



Compile and execute *LabTesta3.java*. Click the buttons, more than once, and observe what happens.

You will see that a button changes colour when you click it. Note that its new colour depends on its previous state. Therefore, the program must remember the state of each button somewhere. You must be careful how this is done.

Look at the code for the main program (*LabTest3a.java*) and note what objects there are:



NOTE: the GUI object **f** *does not hold the state of the 'board' itself*! This separation of interface and data is important. The interface object is only a *window* on to the data which the user is manipulating. Think of the user interface as being like a window through which the user views and manipulates the data objects in the system design.

Using the above figure, track exactly what happens when the user clicks on a button.

Note: the interface object **f** needs to access the state data in **S** (so we pass the state object **S** as a parameter to the interface object **f** at instantiation; **f** thus has full access to the state). But since the state object just responds to calls from **f**, the data from **f** which it needs should be passed as parameters to the method calls. So **S** is not given **f**.

Exercise: Extend the program so that it beeps only when the top left button state goes to true. [using *Toolkit.getDefaultToolkit().beep();*]

4. Movergame (LabMoverGame.java)

This program enables up to two users to compete in moving a ship around a 'sea' containing a couple of islands (black squares). There are two control panels, each of which allows the user to move the ship one place up, down, left or right. Run the program **LabMoverGame.java**, and try out the controls. Observe what happens.

Challenge!! Modify the *LabMoverGame* program so that there is a hidden mine on the grid which, if the ship land on it, explodes! After that, the ship is lost!

Notes: (i) the mined square should be displayed as the same colour as the background.
(ii) the mined square should be set up at initialisation.