THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Electrical Engineering and Computing

Inft2012 Application Programming – Notes for week 2

Non-default event handlers

You normally create an event hander by double-clicking on the control in the design view of the form. For example, by double-clicking on a button, you get its Click event handler; by double-clicking on the form (probably by accident), you get its FormLoad event handler; and so on.

This is generally the only way you should create an event handler. If you just type the code, even exactly as you see it in an example, you're not creating an event handler – because connecting that control to that handler involves a further line of code, which is automatically generated in the Designer code file that you won't often need to look at.

Controls generally have many more events than the default one. For example, you might want your program to do something when the cursor hovers over a text box. Here's what you would do:

- In design view, select the text box for which you want the handler.
- In the Properties window, select Events – the lightning bolt symbol. This gives you a list of all the possible events for that control.
- Scroll down the list until you find the event you want, in this case MouseHover.

Double-click that event. You will be taken to the code window, in the skeleton of the event handler you require.

double, not Double

Remember that C# is case-sensitive. The primitive type we use for non-integer numbers is double, not Double. Double does exist, but it's not a primitive type, and it's slightly different from double.

Caution with picture boxes

When you add a picture box to a form, ensure that its *SizeMode* property is *Normal* – unless you have a good reason to want it some other way. The other modes are set up to stretch either the picture box or the picture in it, and can lead to serious distortions.

Lab exercises

Most students will probably not finish all of these exercises in the lab class. Remembering that learning to program takes lots of practice, you are strongly advised to finish them before next week's lab class, bringing to the class any problems you encounter.

1. Reading and understanding program code is an essential step on the way to writing program code. Read the following code and explain what it does. Try to explain what the purpose of the code might be, rather than its individual steps. Do this just by reading the code (and perhaps making notes or sketches) – *not* by entering it into a program and trying it. The four integer arguments of DrawLine are the x and y coordinates of the start of the line and the x and y coordinates of the end of the line.

```
Graphics graPaper = picbxDrawing.CreateGraphics();
graPaper.FillEllipse(Brushes.White, 50, 100, 100, 100);
graPaper.FillEllipse(Brushes.White, 75, 50, 50, 50);
graPaper.FillEllipse(Brushes.Black, 85, 65, 10, 10);
graPaper.FillEllipse(Brushes.Black, 105, 65, 10, 10);
Pen penBlack = new Pen(Color.Black, 3);
```

```
        graPaper.DrawLine(penBlack, 85, 85, 115, 87);
        graPaper.DrawLine(penBlack, 75, 115, 35, 160);
        graPaper.DrawLine(penBlack, 125, 115, 165, 160);
```

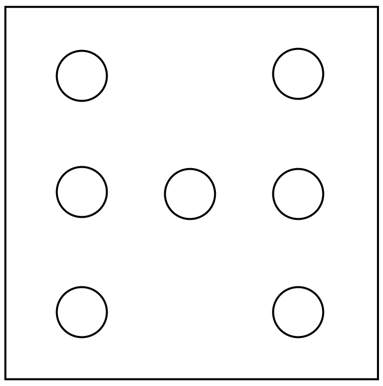
2. Create the Drawing program as illustrated in the lecture, saving it to a sensible location (such as an Inft2012Projects folder on your thumb drive).

3. Once you have the program debugged and running, use Solution Explorer to take another look, a close look, at the Drawing.Designer.cs code. See if any of it makes any sense to you. Can you find anything that says to create a form? A button? A picture box?

4. Back in the frmDrawing.cs code, change the name of the event handler. Change it from btnDraw_Click to any old rubbish. Depending on which option you choose next, the program might still work. Try it and see.

   Now explain what you saw happen. The explanation is reasonably simple, but involves looking in the Drawing.Designer code.

5. Imagine the face of a die (the singular of dice) as being represented by a square with seven places in which small circles can be drawn.



   Create a new project (with a suitably informative name) to display the different faces of a die. The form will have six buttons, each displaying a number from 1 to 6, and a square picture box. When a button is pressed, the program will draw in the picture box the circles that make up that number on the face of the die.

   Most of what you need to know to do this was covered in the week 1 lecture, but you might need one additional method. A Graphics object has a method called Clear, which takes a colour as its argument, and clears the object to that colour. For example, if you have a graphics object called graDie, you might write graDie.Clear(Color.Yellow).

   In case you’re not sure, these are the circles that make up each number:

   1: centre
   2: middle left, middle right
   3: bottom left, centre, top right
   4: top left, bottom left, top right, bottom right
   5: top left, bottom left, centre, top right, bottom right
   6: top left, middle left, bottom left, top right, middle right, bottom right

You should soon discover that this is one of those apparently simple instructions that actually involve quite a lot of thought and work. Even so, you are urged to finish this exercise, as you will later be asked to build on it.

6. Create a Project called MilesToKm, to convert miles to kilometres. You will need a textbox for the miles, a button and some means to display your result (a message box, a label, or another text box).

   Modify the project so that when the cursor enters the miles text box, its background becomes yellow, and when the cursor leaves it, it becomes white again.

7. Create a Project called ButtonProperties and place 3 buttons on the form. Give the form a title of Property Change Demonstration. The text on the buttons should read 1, 2, and 3. Place suitable code statements in the code for the buttons so that:
   - Clicking Button 1 will change all the buttons to read "Press Me";
   - Clicking Button 2 will change the buttons to read 1, 2, and 3 again;
   - Clicking Button 3 will make Buttons 1 and 2 disappear or reappear (using the Visible property of both buttons).

8. Write a program that converts a Fahrenheit temperature to its equivalent Celsius temperature. The formula is:
   dDegreesCelsius = (dDegreesFahrenheit – 32) * 5 / 9
   How will you know whether your program is converting correctly? Check that it is.

9. An initial number of seconds is entered in a text box. Write a program that will convert this to hours, minutes and seconds. (Hint: use the integer division and remainder operators.) *Perform a sample pen and paper solution before you begin the program, to work out the logic and sequence of operations involved.* If you can't work out how to solve the problem, don't start writing the program. Instead ask your tutor for help.

   Use a single message box to display the result as:
   ```
   Hours: 1     Minutes: 24   Seconds: 9
   ```
   (which is the correct answer for 5049 seconds)