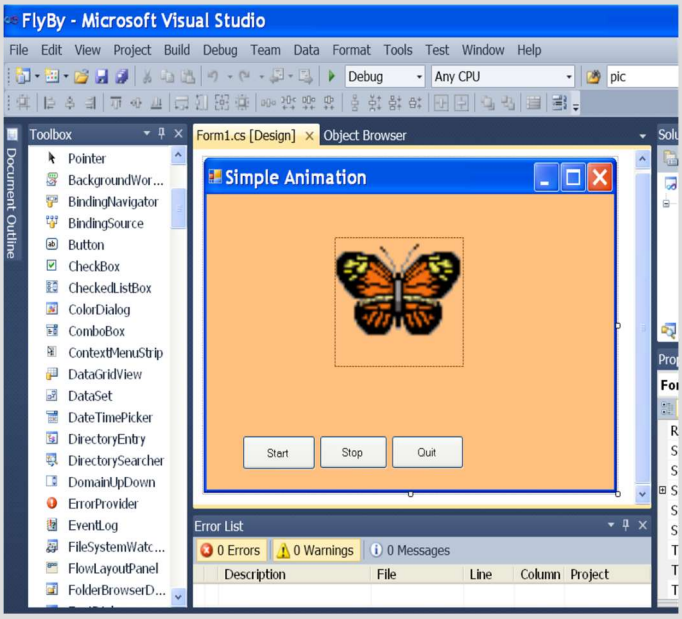
Windows Programming

bucks new university

.NET Windows Programming Using C#



Application Programming (CO453)

Part C – Weeks 9-13

C# Windows Programming
Unit 6

Graphics

Introduction to Graphics

The .NET framework contains thousands of classes and methods. For convenience these are divided into "namespaces" which contain sets of related classes.

- For all graphics programs the **System.Drawing** namespace is required
- so include the line: **using System.Drawing;**
- **System.Drawing** is the 'home' of the **Graphics** class which contains drawing methods such as **DrawString()**, **DrawLine()**, **DrawRectangle()**, etc.
- In order to use these methods we must first create a new Graphics object.
- Note: Every form has a **Paint()** method that is automatically called when some event causes the form to be repainted. We shall be using this Paint() method extensively during this graphics section.

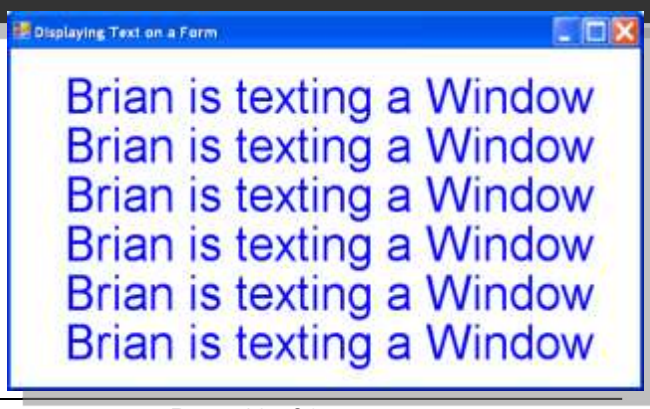
6.1 Using DrawString()

- Open the existing Windows application called **Graphics1**
- Run this and see how it uses **DrawString()** to draw large blue text on the screen in the **Form1_Paint()** method

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
        int  x = 50;                          // set the horizontal x coordinate
        int  y = 20;                          // set the vertical y coordinate
        Graphics g = e.Graphics;              // get a graphics object g
        Font myFont = new Font("Arial", 20);  // create a new size 20 font
        System.Threading.Thread.Sleep(1000); // pause for 1 second
          // now draw text using myFont and a blue brush at position x, y
        g.DrawString("Brian is texting a Window", myFont, Brushes.Blue, x, y);
}
```

Task 6.1: DrawString()

1. Change any references to **Brian** to use your own name
2. Change the form **BackColor** to Yellow
3. Change the text colour to Red
4. Choose a different Font style (instead of Arial) and increase its size to 30
5. Now use a **for loop** to do the following:
   a. Draw the string 6 times
   b. Each line should be underneath the previous one
   c. There is a delay of 0.5 seconds between each one
6. Minimise the window and then restore it again .. explain what happens
7. Add the line **g.Clear(BackColor);** to the loop and explain what happens



6.2 Drawing Rectangles and Ellipses

- Open the existing Windows application called **Graphics2**
- Run it and see how it uses **DrawRectangle()** to draw a blue outline and **FillRectangle()** to fill the rectangle with a red colour
- Notice also that a white BackColor has been set in the Form1_Load() method:
  this.BackColor = Color.White;
- Now look at the code in the Form1_Paint() method:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    int x = 80,  y = 10;                          // x, y position of Rectangle
    int w = 300, h = 200;                         // width and height of Rectangle
    Graphics g = e.Graphics;                      // get a graphics object
    Pen myPen = new Pen(Color.Blue, 10);          // define a new blue pen, size 10

    g.DrawRectangle(myPen, x, y, w, h);           // draw a rectangle using pen
    g.FillRectangle(Brushes.Red, x, y, w, h);     // fill the rectangle with red
}
```

Note: Creating a Pen for Drawing

- **DrawRectangle()** draws an outline .. a **pen** must be first set up for it to use.
- Here a new pen called **myPen** has been defined using the colour Blue and a thickness of 10 pixels has been set for the pen.
  Pen myPen = new Pen(Color.Blue, 10);

Note: DrawRectangle(pen, x, y, width, height)

- **DrawRectangle()** uses a pen to draw at the coordinates **x, y** on the form.
- The last 2 parameters are the **width** and **height** of the rectangle

Note: FillRectangle(brush, x, y, width, height)

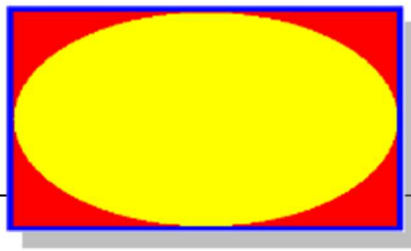
- **FillRectangle()** uses a brush colour rectangle at the coordinates **x, y** on the form. The last 2 parameters are the **width** and **height** of the rectangle

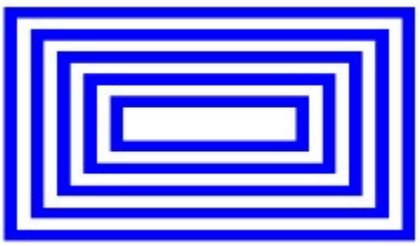
Note: DrawEllipse() and FillEllipse()

- These work in a similar way to **DrawRectangle()** and **FillRectangle()**

Task 6.2: Rectangles and Ellipses

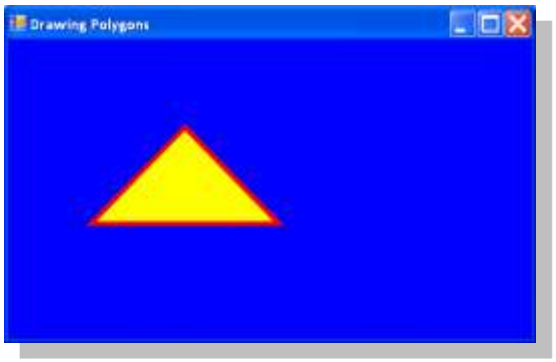
1. Add code that uses **FillEllipse()** to draw a yellow ellipse, the same size and position as the rectangle .. look at the result (see below)
2. Add code that uses a **for loop** to draw 6 rectangles (use **DrawRectangle()**) so that each rectangle appears inside the previous one (see below).
   - You will need to change the values of x, y, w and h within the loop





6.3 Drawing Polygons

- Open the existing Windows application called **Graphics3**
- Run the application and see the result:
- A polygon can have any number of sides .. this one has 3
- The program uses **DrawPolygon()** and **FillPolygon()** to draw a 3 sided polygon (a triangle!)
- Look at the code in the **Form1_Paint()** method:




```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;                    // get a graphics object
    Pen myPen = new Pen(Color.Red, 10);         // create a new red pen

    Point[] shape = new Point[3];               // create an array of points
    shape[0] = new Point(200,100);              // put 3 points into the array
    shape[1] = new Point(300,200);
    shape[2] = new Point(100,200);

    g.DrawPolygon(myPen, shape);                // draw shape using red pen
    g.FillPolygon(Brushes.Yellow, shape);       // fill shape using a yellow brush
}
```

Notes

- DrawPolygon() and FillPolygon() both use a shape object which is an array of **Point**
- 3 new points have been added to the shape object (these are the x, y coordinates of the 3 points of the triangle)

Task 6.3: Drawing Polygons

1. Design a hexagonal shaped polygon with 6 points .. it does not have to be perfectly symmetrical .. you may find it is a good idea to plan this on paper first to work out the coordinates!
2. Change the background colour to **Yellow**
3. Make the necessary changes to fill a **red** hexagon using your design
4. Underneath the hexagon use **DrawString()** to print a message using your name

Task 6.4: Pick a Graphic

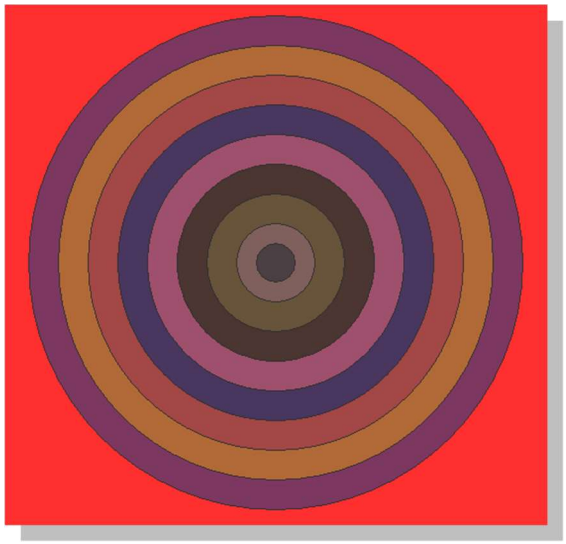
Design a program that has the following features:

1. Pressing the **L** key produces a **Line** at a random position in a random colour
2. Pressing **R** produces a random **Rectangle** in a random colour
3. Pressing **E** produces a random **Ellipse** in a random colour
4. Pressing **C** produces a random **Circle** in a random colour
5. Pressing **S** produces a random **Square** in a random colour
6. Pressing **M** produces a match stick figure.

Task 6.5: Concentric Circles

Design a program that does the following :

- When the program runs, a series of 10 concentric circles using random colours is produced, decreasing in size.



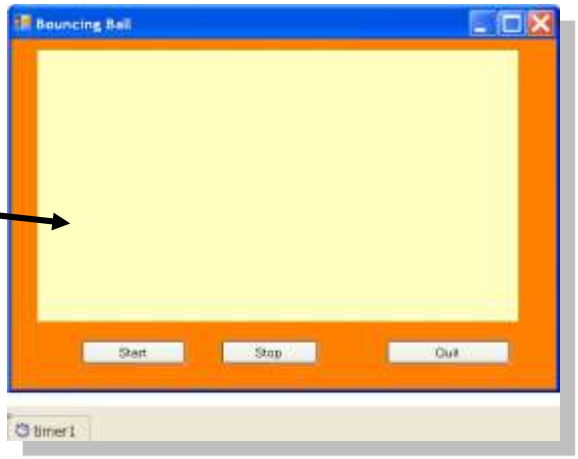
A note on Random RGB Colours

- We can set rgb colours (red, green, blue) using **Color.FromArgb()**
- For example: **this.BackColor = Color.FromArgb(50, 50, 50);**
- Each of the 3 colour components range from 0 to 255 so we can set a random values using:

```
Random  r = new Random();      // create a random generator object r
int red = r.Next(256);         // pick a random value for red (etc.)
this.BackColor = Color.FromArgb(red, green, blue);
```

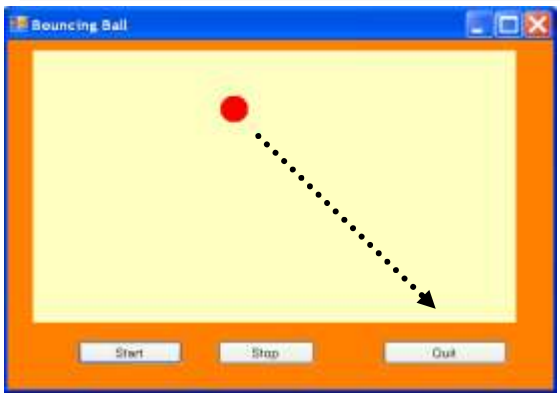
Unit 6: Independent Study

6.6 Bouncing Ball



- Open the existing Windows application called **Ball**
- Note that it has:
  - a PictureBox (**pbxDisplay**)
  - A **timer** that will animate the display
  - 3 **buttons** to **start** the timer, **stop** the timer and **Quit** the application

- Run the application and see a **red ball** in the PictureBox .. if you now click the **Start** button it moves off the screen.
- What we need to do is get the ball to bounce around inside the PictureBox



Looking at the Code

- If you look at the code you will see that the ball is being drawn as a red circle by using **FillEllipse()** in the **pbxDisplay_Paint()** method:

```
private void  pbxDisplay_Paint(object sender, PaintEventArgs e)
 {
        Graphics g = e.Graphics;      // get a graphics object
         // draw a red ball, size 30, at the position set by x and y
        g.FillEllipse(Brushes.Red, x, y, 30, 30);
}
```

- The **timer** is used to change the position of the ball by increasing the value of x and y (see code below). This is done each time the timer ticks.
  Note: **xmove** and **ymove** variables have been initialised to **10** earlier.
- The timer also calls the **Refresh()** method .. which causes **pbxDisplay_Paint()** to be called again .. and so the ball is drawn again at the new x, y position.

```
private void  timer1_Tick(object sender, EventArgs e)
{
        x += xmove;   y += ymove;        // add 10 to the x and y positions
        Refresh();                       // refresh the screen .. calling Paint()
}
```

- Task: now you must **add some more code** to the **timer1_Tick()** method to get the ball to bounce when it reaches the bottom edge of the picturebox:
  - Note: 30 is the size of the ball

```
if  (y + 30  >=  pbxDisplay.Height)       // is ball position at bottom of picturebox?
{
        ymove =  -ymove;                  // if so, set the opposite movement
}
```

Task 6.6: Bouncing Ball

1. Get the ball to bounce around continually when it reaches any of the edges of the **pbxDisplay** picturebox

6.7 Detecting a KeyPress

- Open the Windows Application called **TestKeys**
- Run the program and press a selection of keys .. including the **'B'** key.
- Look at the code and find a special method called **ProcessCmdKey()** .. this method allows us to **override** the normal processing of key presses.
- You will see that at the moment the program only responds to the key **B**
- Modify the code to respond to the **Up** key and the **Down** key.

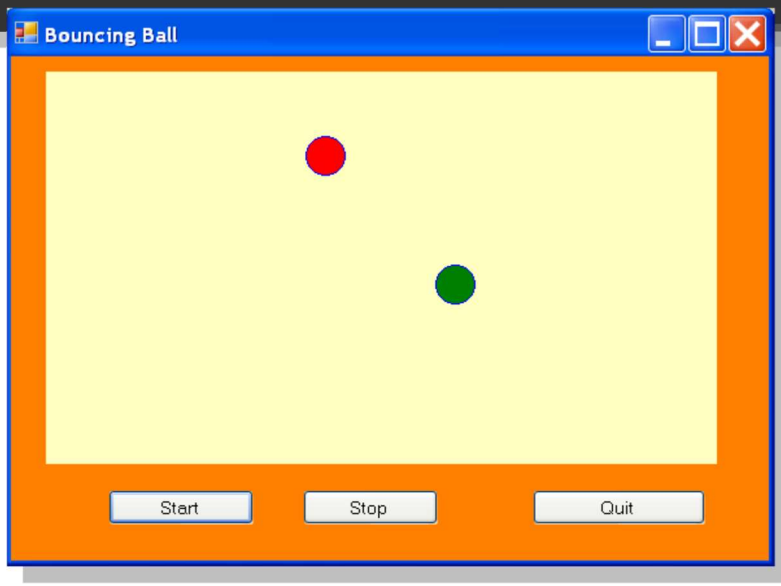
```
protected  override  bool  ProcessCmdKey(ref Message msg, Keys keyData)
{
        string input;
        input = keyData.ToString();              // collect the key data
        if  (input == "B")                       // if 'B' key pressed
        {
              MessageBox.Show("You pressed B" );
              return true;
         }

        return false;                  // return false if no key processed
}
```

Task 6.7: Bigger Balls

1. To save you some typing, copy the **ProcessCmdKey()** method from the **TestKeys** project and paste it into the previous **Ball** project code.
   - Just test the project again to see that it still works OK and that the Up and Down key presses can now be detected by the new method.
2. Modify the program so that pressing the **Up** key keeps making the ball **Bigger** .. while pressing the **Down** key makes the ball **Smaller**.
   - **Hints**: you will need to set a variable for the **size** of the ball and use this variable in the **FillEllipse()** method
   - Then you can increase the size or decrease the size in the appropriate part of the ProcessCmdKey() method
   - Remove the MessageBox() instructions now as you no longer need them

Task 6.8: More Balls

1. Add code to your previous project so that there are **2 balls** .. each one starting from a different position .. both balls move around and bounce off the walls.
2. Get the 2 balls to start from random x and y positions.
3. Add code so that the 'C' key can be used to change the BackColor of the picturebox.
4. Tricky: Can you devise a way to get the balls to bounce when they hit each other?



Task 6.9: Drawing Skills

1. Design a picture of a house with roof, windows, door, chimney etc.
2. Write a program using the tools available to you to produce your drawing when the program runs

Your Log Book

- Your logbook should include the following for all the Graphics exercises:
  - Headings and task summaries
  - Screen shots of the running programs
  - Code taken from the **Form1.cs** commented with your **name**, **date** and **project details**