## GRAPHICS OBJECT

- You need to tell Visual C# Express that you will be using graphics methods with the panel control. To do this, you convert the panel control to something called a **graphics object**. Graphics objects provide the "surface" for graphics methods. Creating a graphics object requires two simple steps. We first declare the object using the standard declaration statement. If we name our graphics object **myGraphics**, the form is:

```
Graphics myGraphics;
```

- This declaration is placed in the **general declarations** area of the code window, along with our usual variable declarations.
- Once declared, the object is created using the **CreateGraphics** method:

```
myGraphics = controlName.CreateGraphics();
```

- **ControlName** is the name of the control hosting the graphics object (in the blackboard project, the **Name** property of the panel control). We will create this object in the form **Load** event of our projects.

- Once a graphics object is created, all graphics methods are applied to this newly formed object.
- Hence, to apply a graphics method named **GraphicsMethod** to the **myGraphics** object, use:

```
myGraphics.GraphicsMethod(Arguments);
```

- **Arguments** are any needed information by the graphics method.

- There are two important graphics methods. First, after all of your hard work drawing in a graphics object, there are times you will want to erase or clear the object. This is done with the **Clear** method:

```
myGraphics.Clear(Color);
```

- This statement will clear a graphics object (myGraphics) and fill it with the specified **Color**.
- The usual color argument for clearing a graphics object is the background color of the host control (controlName), or:

```
myGraphics.Clear(controlName.BackColor);
```

- Once you are done drawing to an object and need it no longer, it should be properly disposed to clear up system resources. To do this with our example graphics object, use the **Dispose** method:

```
myGraphics.Dispose();
```

- This statement is usually placed in the form **FormClosing** event method.

- Our Blackboard drawing will require colors and objects called pens, so let's take a look at those concepts. Doesn't it make sense we need pens to do some drawing?

## COLORS

- You can define variables that take on color values. It is a two step process. Say we want to define a variable named **myRed** to represent the color red. First, in the general declarations area, declare your variable to be of type **Color**:

```
Color myRed;
```

- Then, define your color in code using:

```
myRed = Color.Red;
```

- From this point on, you can use **myRed** anywhere the red color is desired.

## PEN OBJECT

Many of the graphics methods (including the method to draw lines) require a Pen object. This virtual pen is just like the pen you use to write and draw. You can choose color and width. You can use pens built into C# or create your own pen.

- In many cases, the pen objects built into Visual C# are sufficient. These pens will draw a line 1 pixel wide in a color you choose (Intellisense will present the list to choose from). If the selected color is ColorName (one of the 141 built-in color names), the syntax to refer to such a pen is:

```
Pens.ColorName
```

- Creating your own pen is similar to creating a graphics object, but here we create a **Pen** object. To create your own Pen object, you first declare the pen using:

```
Pen myPen;
```

- This line goes in the general declarations area. The pen is then created using the Pen constructor function:

```
myPen = new Pen(Color, Width);
```

- **Color** is the color your new pen will draw in and **Width** is the integer width of the line (in pixels) drawn. The pen is usually created in the **form Load** method. This pen will draw a solid line. The **Color** argument can be one of the built-in colors or one generated with the **FromArgb** function.

- Once created, you can change the color and width at any time using the **Color** and **Width** properties of the pen object. The syntax is:

```
myPen.Color = newColor;
myPen.Width = newWidth;
```

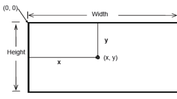
- Here, **newColor** is a newly specified color and **newWidth** is a new integer pen width.
- Like the graphics object, when done using a pen object, it should be disposed using the **Dispose** method:

```
myPen.Dispose();
```

- This disposal usually occurs in the form **FormClosing** method.

## GRAPHICS COORDINATES

- We will use Visual C# Express to draw lines using a method called the **DrawLine** method. Before looking at this method, let's look at how we specify the points used to draw and connect lines. All graphics methods use a default **coordinate system**. This means we have a specific way to refer to individual points in the control (a panel in our work) hosting the graphics object. The coordinate system used is:




- We use two values (coordinates) to identify a single point in the panel. The x (horizontal) coordinate increases from left to right, starting at 0. The y (vertical) coordinate increases from top to bottom, also starting at 0. Points in the panel are referred to by the two coordinates enclosed in parentheses, or (x, y). Notice how x and y, respectively, are similar to the Left and Top control properties. All values shown are in units of pixels.

### DRAWLINE METHOD

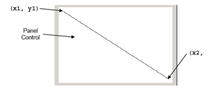
- The Visual C# **DrawLine** method is used to connect two points with a straight-line segment. It operates on a previously created graphics object. If that object is **myGraphics** and we wish to connect the point **(x1, y1)** with **(x2, y2)** using a pen object **myPen**, the statement is:

```
myGraphics.DrawLine(myPen, x1, y1, x2, y2);
```

- The pen object can be either one of the built-in pens or one you create using the pen constructor just discussed. Each coordinate value is an integer type. Using a built-in black pen (**Pens.Black**), the **DrawLine** method with these points is:

```
myGraphics.DrawLine(Pens.Black, x1, y1, x2, y2);
```

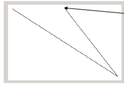
- This produces on a panel (**myGraphics** object):




- To connect the last point (x2, y2) to another point (x3, y3), use:

```
myGraphics.DrawLine(Pens.Black, x2, y2, x3, y3);
```

- This produces on a panel (myGraphics object):




- For every line segment you draw, you need a separate **DrawLine** statement. To connect one line segment with another, you need to save the last point drawn to in the first segment (use two integer variables, one for x and one for y). This saved point will become the starting point for the next line segment. You can choose to change the pen color at any time you wish. Using many line segments, with many different colors, you can draw virtually anything you want!