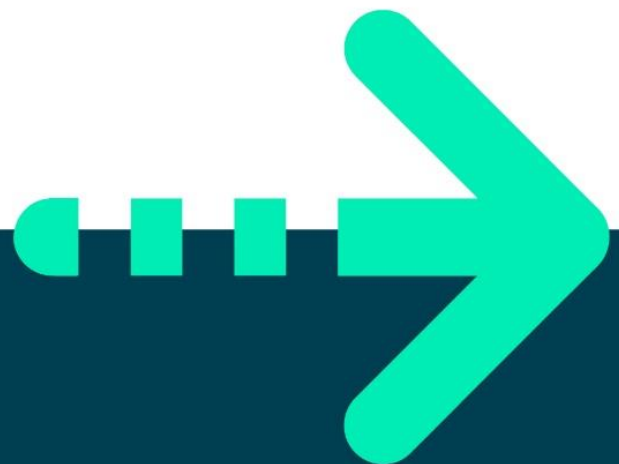




LAB 9, C# TYPES II – FIELDS AND METHODS

FUNDAMENTALS



Lab 9, C# - Types II – Fields and Methods

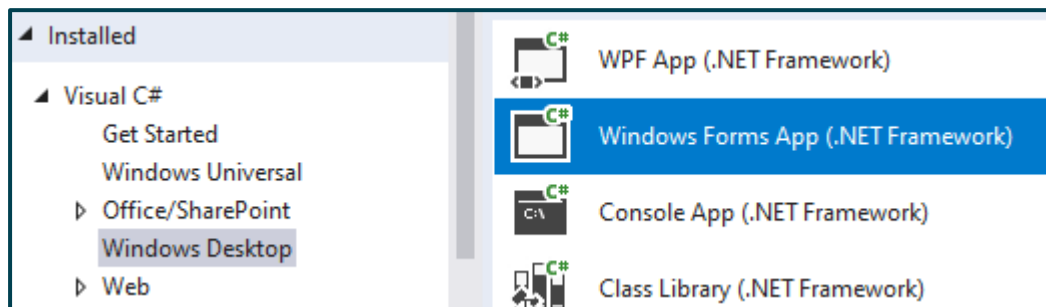
Objective

You will further your understanding of object oriented programming. In this lab you'll create a graphical application instead of a Console app.

Part 1


You'll soon be creating a graphical application which is new. You'll copy and paste many lines of code. You are not required to understand how the graphics code works but it is not too complex to investigate by yourself.

1. Create a new Windows Forms App called **Lab09**



2. Have a look at the **Program** class. It creates an instance of a class called **Form1** which represents a Windows Form. You'll use this to display the graphical objects.
3. You need a canvas upon which you can draw and show your animation. In this case you'll create a number of ball objects bouncing inside a rectangle (the world).

You can get creative later and do something else but for now please follow the steps. The aim is to investigate OO not concentrate on animation and graphics.

4. Click on the body of the Form and then press F4 to see its properties (if not displayed already)
5. Click the lightening symbol  to see the form's events. An **event** is an action which you can handle in code. For example, when a user clicks on a control, the control's click event is fired. In this application, we're after the **Paint event** which is fired when the form needs to be redrawn or repainted.
6. Double-click on the **Paint** event from the list of events. You'll then be redirected to the code behind the form.
7. Type the following code in the Paint event. This code clears the screen and draws a red rectangle. 10,10,600 and 400 is the world's dimensions.

```
private void Form1_Paint(object sender, PaintEventArgs e)
```

```
{
    e.Graphics.Clear(Color.Gray);
    e.Graphics.FillRectangle(Brushes.Red, 10, 10, 600, 400);
}
```

8. Run the application to make sure everything works so far.
9. Stop running by closing the Form.
10. Double-Click on the Form1 icon in the Solution Explorer window to view it in design mode.

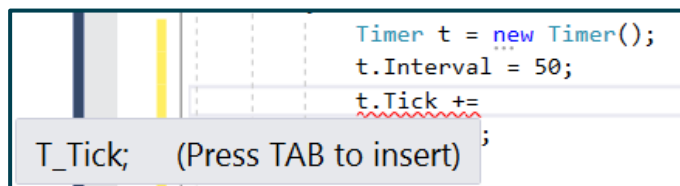
11. Press **F4** and then click  to see the form's events.

12. Double click on the **Load event** from the list to view an event which fires when the form is loaded for the first time.

Type the following code in the *Load event* in order to create a timer which fires every 50 millisecond.

```
private void Form1_Load(object sender, EventArgs e)
{
    Timer t = new Timer();
    t.Interval = 50;
}
```

13. Add the following text to the above method **t.Tick +=** and then press the Tab key to create an event handler for the Timer's tick event.

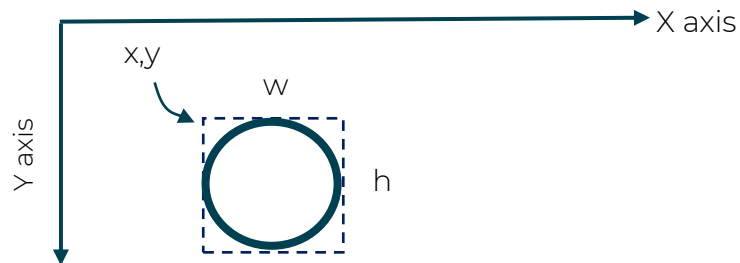


14. In the Tick event type: **this.Invalidate();**
This will force the *Paint event* to fire every 50ms.
15. Back in the *Form Load* event, activate the timer by adding this line of code:
t.Start();
From now all your code for drawing will go into the Paint event.

Create the Ball class

16. Create a class called **Ball** with the following fields

```
public int x, y, w, h;
private int dirX, dirY;
```



x and y represent the top coordinate of the shape (ball) within an upside down coordinate system. dirX and dirY are the amount by which a ball moves in the x and y directions.

17. Create a constructor to set every field (x,y,w,h,dirX, dirY).
18. To practice constructor chaining, create another constructor to set all the fields except dirX and dirY.
- Using constructor chaining, call the first constructor and pass it dirX=1 and dirY=1.
19. Create a method called **move()** in the Ball class.
- In this method, increase x by dirX and y by dirY and then follow a series of tests to make sure the ball moves within a designated world area (x=10, y=20, width=600, height=400) which you set in step 8.
- if (x < 10) then set x=10 and change the sign of dirX
Tip: dirX = -dirX.
Do the same for the y coordinate(changing dirY of course).
- if (x > 600 – the width of the ball) then set
x=600 – the width of the ball and reverse the sign of dirX.
Do the same for the y coordinate taking the ball's height into account.
20. Create an array field (at the beginning of the class Form1) as: **Ball[] balls;**
21. Back in the Form Load event handler, create three instances of Ball and place them inside the **balls** field.
22. In the *Paint event*, create a foreach loop to go through each ball and run its **move()** method. You will then draw the ball after moving it. Please use the code below to draw a ball:
- ```
e.Graphics.DrawEllipse(Pens.Yellow, b.x, b.y, b.w, b.h);
//where b is the ball reference you created earlier
```



23. If everything works, you should see all 3 objects moving around within the world's rectangle.

If you have a need for speed then increase the values held by dirX and dirY!

## Part 2

In this part you'll make use of the static keyword.

Clearly setting the world dimension to x=10, y=10, w=600, h=400 is not a good idea. All balls should share the same world coordinates. This is a perfect place to use the static keyword.

1. Create four **static int** fields in the Ball class called worldX, worldY, worldW and worldH to represent the x, y, width and height of the world.
2. Create static method to set the static fields for the world coordinates as **public static void setWorld(int wx, int wy, int ww, int wh)**
3. Modify the Ball class code to use the world's coordinates.
4. At the beginning of the Form Load event Invoke the setWorld() method of the Ball class to set the world's coordinate for every Ball.
5. Run your code to see if it works.

**\*\* End \*\***

