C# .Net Events

In programming terms, an Event is when something special happens. Inbuilt code gets activated when the event happens. The event is then said to be "Handled". The Events that we'll discuss in this section are called GUI Events (GUI stand for Graphic User Interface). They are things like clicking a mouse button, leaving a text box, right clicking, and many more. We'll start with the Click event of buttons.

The Click Event for C# Form Buttons

The click event gets activated when a button is clicked on. Examine the default code for a button:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

In between the round brackets, we have this:

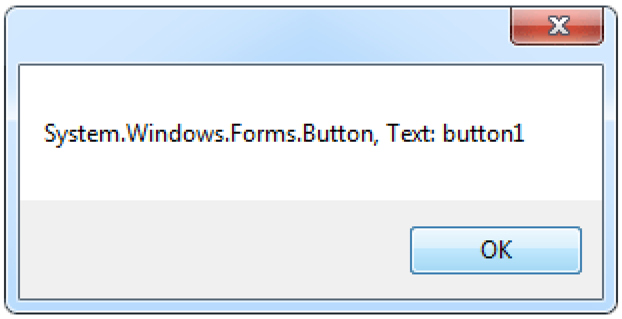
object sender, EventArgs e

The object keyword refers to the object which activated the event, a button in this case. This is being placed in a variable called sender. You can test this for yourself.

Start a new project. Add a button to your new form and double click it. Place the following code between the curly brackets:

```
MessageBox.Show( sender.ToString() );
```

We're just using the ToString method on the sender variable. Run your programme and click the button. You should see this:



```
System.Windows.Forms.Button, Text: button1
```

The Message is displaying which object was the sender of the event, as well as displaying the Text property of the sender: the button with the Text "button1".

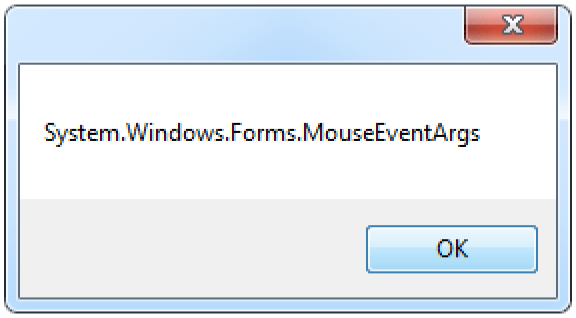
The other argument in between the round brackets was this:

EventArgs e

EventArgs is a class. It's short for event arguments, and tells you which events was raised. The letter "e" sets up a variable to use this class. If you change your line of code to this:

MessageBox.Show(e.ToString());

the message box will then display the following:



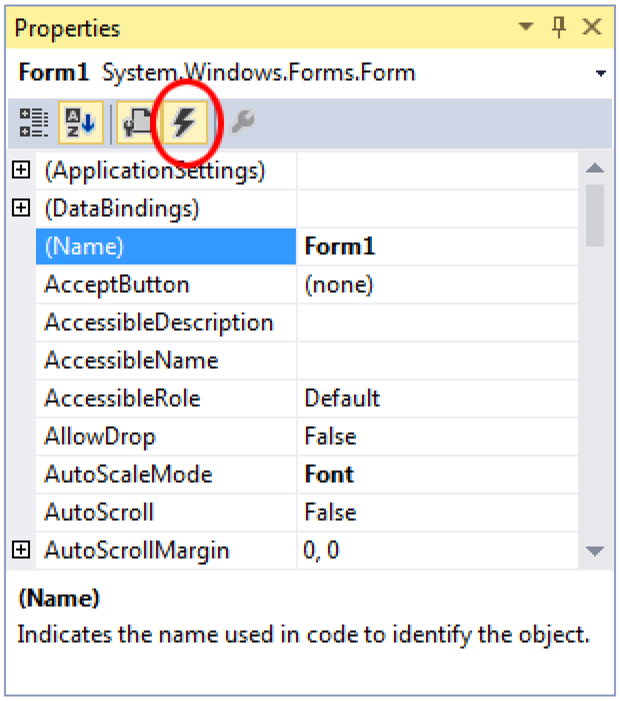
```
System.Windows.Forms.MouseEventArgs
```

So clicking raises a Mouse Event Argument. C# .NET already knows what to do with an event of this kind, so you don't need to write any special code yourself. But what if you wanted to know which button was clicked, the left button or the right?

The Mousedown Event In C# .Net

You can see what event are available for a particular control in the Properties area.

In Design View, click on your Form1 to select it, instead of the button. To see what events are available for the Form itself, click the lightning bolt at the top of the Properties area, as in the image below:

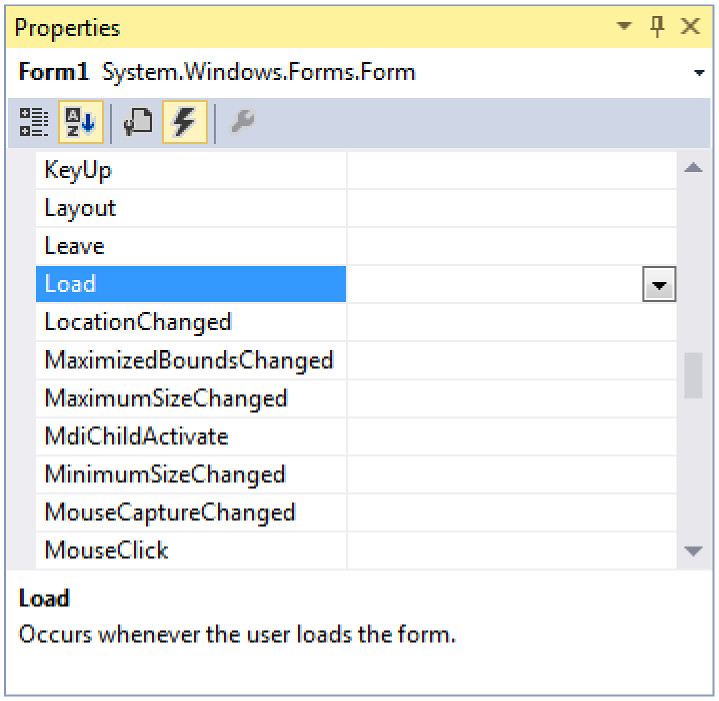

Properties

Form1 System.Windows.Forms.Form

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	Form1
AcceptButton	(none)
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScaleMode	Font
AutoScroll	False
⊞ AutoScrollMargin	0, 0

(Name)
Indicates the name used in code to identify the object.

When you click the lightning bolt, you'll see a list of events appear:


Properties

Form1 System.Windows.Forms.Form

KeyUp	
Layout	
Leave	
Load	
LocationChanged	
MaximizedBoundsChanged	
MaximumSizeChanged	
MdiChildActivate	
MinimumSizeChanged	
MouseCaptureChanged	
MouseClick	

Load
Occurs whenever the user loads the form.

You've already met the Load event, so we won't cover it here. But notice how many events there are.

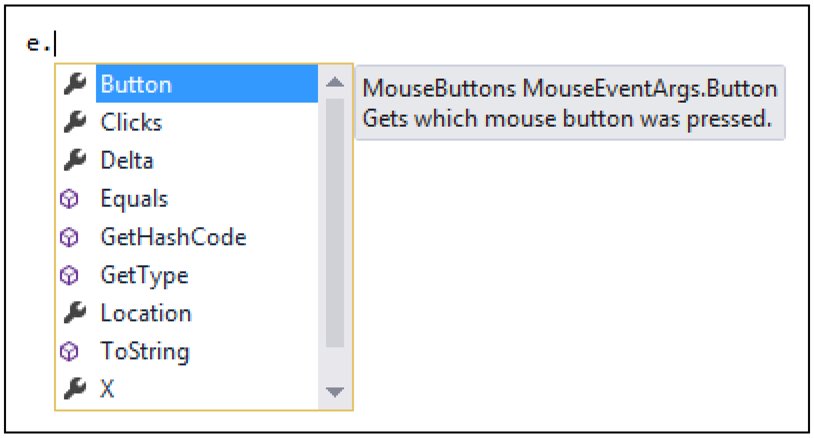
Locate the MouseDown event on the list. Now double click the word "MouseDown". You should see a code stub appear:

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    |
}
```

In between the round brackets, there is still a sender object. But notice the new argument:

MouseEventArgs e

The letter "e" is the default variable name. The type of variable belongs to the MouseEventArgs. You can see what this does by typing the letter "e", then a full stop (period). You should see the IntelliSense list appear:



```
e.|
```

MouseButtons MouseEventArgs.Button
Gets which mouse button was pressed.

The list is displaying the properties and methods available to the e variable. One of these is the Button property.

We can use an if statement to check which mouse button was clicked. Add this to your code:

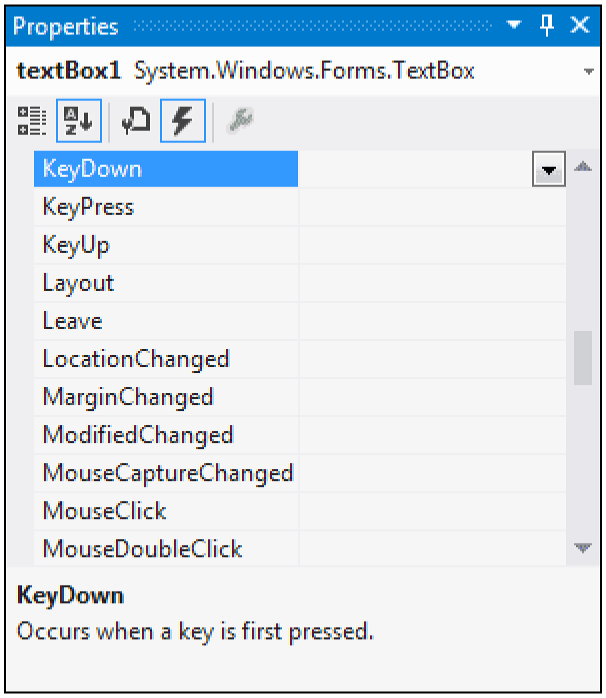
```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        MessageBox.Show("Left button clicked");
    }
    else if (e.Button == MouseButtons.Right)
    {
        MessageBox.Show("Right button clicked");
    }
}
```

Run your programme and click either of your mouse buttons on the form. You should see a message box display.

The Keydown Event In C# .Net

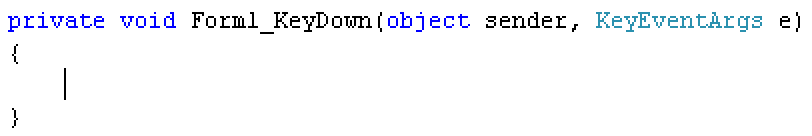
The KeyDown event works in a similar way the MouseDown event - access an EventArg and write code to detect which key was pressed. But a KeyDown event for a form doesn't make much sense. So add a textbox to your form.

From the Properties area on the right, locate the KeyDown events for your textbox:



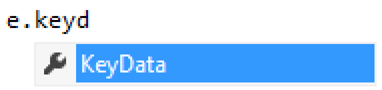
KeyDown
Occurs when a key is first pressed.

Now double click the name of the event to open up its code stub. You should see this:



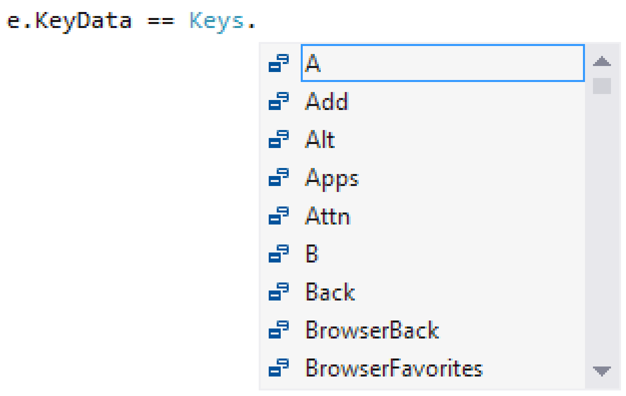
```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    |
}
```

This time, instead of MouseEventArg we have a **KeyEventArgs**. After typing a letter "e", you'll see the IntelliSense list display a list of available options.
Select **KeyData** from the list:



```
e.keyd
```

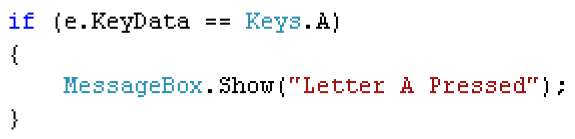
The one you use to get at a particular key is the **KeyData** property. After a double equals sign, you then choose **Keys** from the IntelliSense list. Type another full stop and you'll see this list:



```
e.KeyData == Keys.
```

The list contains all the letters, characters and keys from your keyboard.

Wrap all this up in an IF statement and you'll have this:



```
if (e.KeyData == Keys.A)
{
    MessageBox.Show("Letter A Pressed");
}
```

Run your form and test it out. Click inside your textbox. You should see the message box appear when you press the letter "a" on your keyboard.

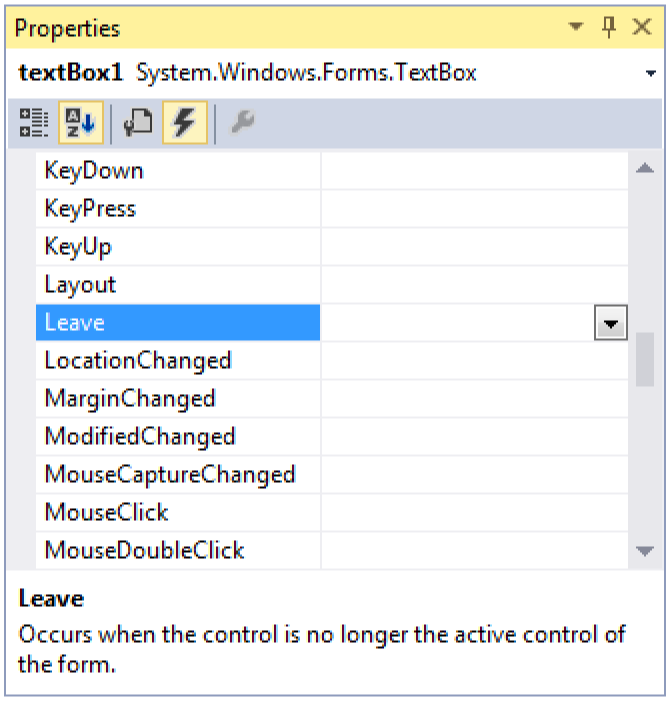
The Leave Event In C# .Net

A very useful event you can use for text boxes is the Leave event. It allows you to validate a text box when a user tries to leave it. You can check, for example, if the text box is blank. If it needs to be filled in, you can keep them there. Let's try that.

Add two text boxes to your form. Locate the TabIndex property for textbox1 and set it to 0. For textbox2, set the TabIndex property to 1.

The TabIndex property refers to which control will be selected when the Tab key is pressed. On a form that a user has to fill in, you want the cursor to jump to the next textbox. You don't want the cursor jumping down from textbox1 to textbox8 - it has to be a nice sequential order. The TabIndex allows you to set the Tab order for all the controls on your form.

With TextBox1 selected, click the lightning bolt in the properties area to see a list of events for textboxes. You should see this:



KeyDown	
KeyPress	
KeyUp	
Layout	
Leave	
LocationChanged	
MarginChanged	
ModifiedChanged	
MouseCaptureChanged	
MouseClick	
MouseDoubleClick	

Leave
Occurs when the control is no longer the active control of the form.

Double click the Leave event to bring up its code stub. Now enter the following code:

```
private void textBox1_Leave(object sender, EventArgs e)
{
    if (textBox1.Text == "")
    {
        MessageBox.Show("You can't leave this box blank");

        textBox1.Focus();
    }
}
```

The code just checks for a blank text box. But it makes the check when the user attempts to leave the text box and move on to the next one. Notice how the user is brought back to the text box:

textBox1.Focus( );

The Focus method can be used to force a control to be selected. For text boxes, this means that the cursor will be flashing inside of it.

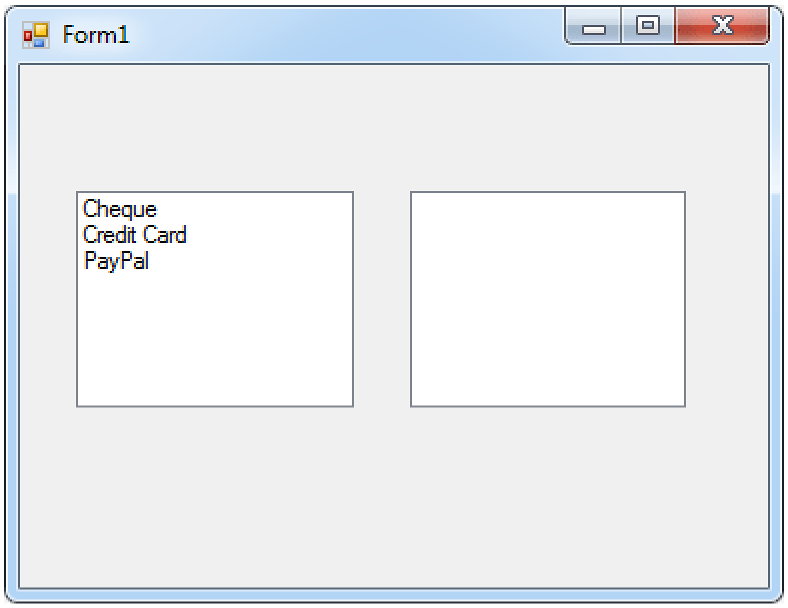
You can also do things like converting text to upper or proper case when the users Tabs away from a text box. Here's some code that converts to uppercase:

```
private void textBox1_Leave(object sender, EventArgs e)
{
    if (textBox1.Text == "")
    {
        MessageBox.Show("You can't leave this box blank");
        textBox1.Focus();
    }
    else
    {
        textBox1.Text = textBox1.Text.ToUpper();
    }
}
```

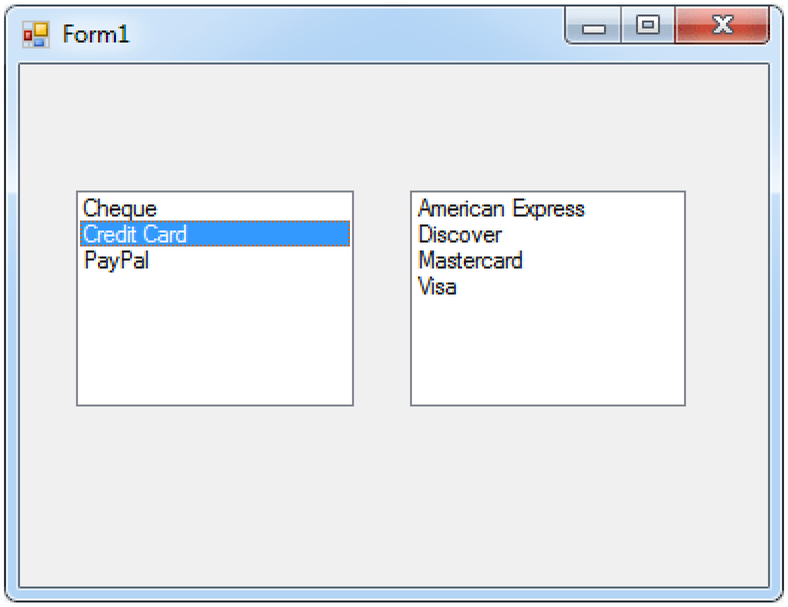
In the next part, we'll take a look at Events for ListBoxes and ComboBoxes.

C# .Net - Listbox And Combobox Events

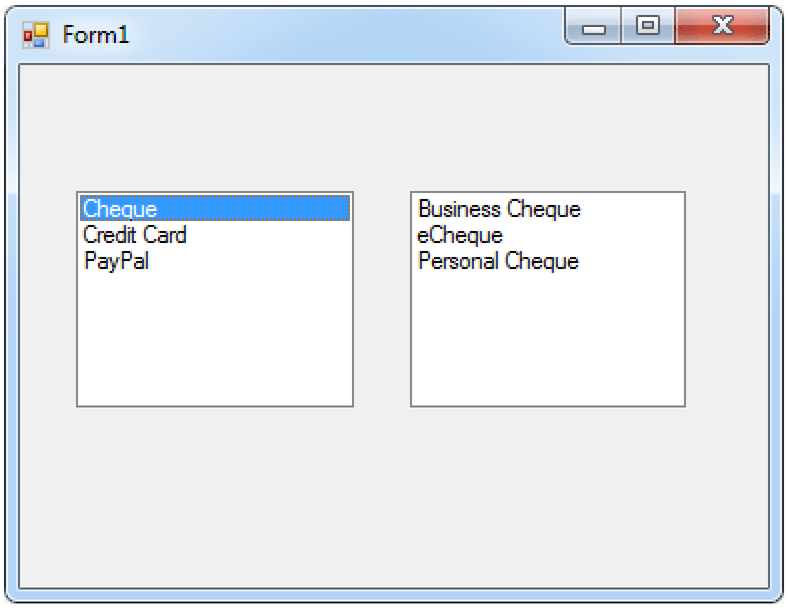
A useful event you may want to use for a ListBox or a ComboBox is the **SelectedIndexChanged** event. You can use this event to get which item in your List or Combo box was selected. The event will fire again when you select a new item. Why is this useful? Well, as an example, you can populate a second ListBox or ComboBox based on which item was selected in the first one. Examine the image below:



We have two ListBoxes on a form. The first one shows payment methods, and the second one is blank. When Credit Card is clicked in the first ListBox, notice what happens to the second ListBox:

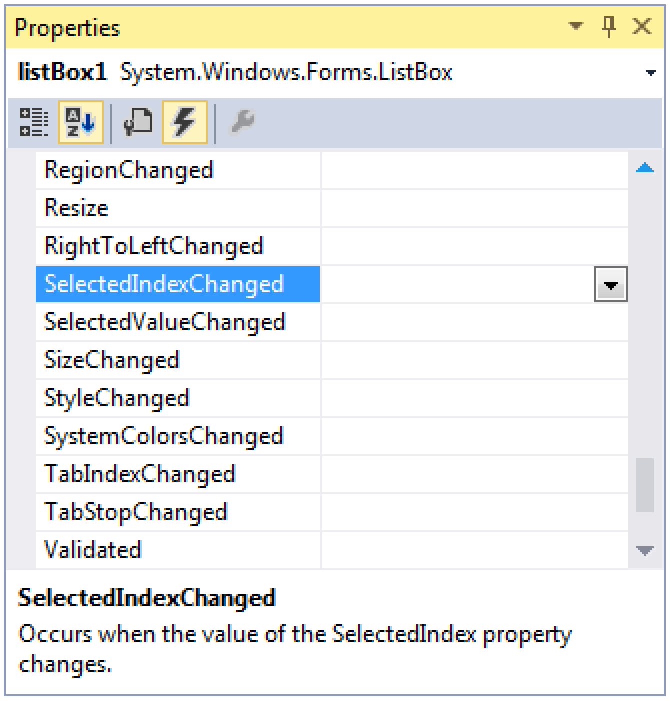


The second ListBox is now displaying the types of credit cards accepted. Click on the Cheque item, and different options are available:



All this happens with the SelectedIndexChanged event.

To see how the code works, start a new project and put two ListBoxes on your form. Click on the first one to select it, and use the Item property to add the three items: Cheque, Credit Card, and PayPal. Now click on the lightning bolt to display the event for a listbox:



RegionChanged
Resize
RightToLeftChanged
SelectedIndexChanged
SelectedValueChanged
SizeChanged
StyleChanged
SystemColorsChanged
TabIndexChanged
TabStopChanged
Validated

SelectedIndexChanged
Occurs when the value of the SelectedIndex property changes.

Double click the SelectedIndexChanged event to bring up its code stub:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    |
}
```

To get at which item in a ListBox or ComboBox was selected, you use the SelectedIndex property. The first item in the list is item 0, the second item 1, etc. So add this if statement to your code:

```
if (listBox1.SelectedIndex == 1)
{

    loadListBox();

}
```

So we're just checking if the selected index has a value of 1. If it does, we're calling a method - loadListBox. Add the method to your code:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox1.SelectedIndex == 1)
    {
        loadListBox();
    }
}

private void loadListBox()

{
    listBox2.Items.Clear();

    listBox2.Items.Add("American Express");
    listBox2.Items.Add("Discover");
    listBox2.Items.Add("Mastercard");
    listBox2.Items.Add("Visa");
}
```

You've already covered ListBoxes in an earlier section, so we won't explain how it works. But Run your programme and test it out. You should find that it works as described above.

Exercise N
Add a second method that loads items when the Cheque payment method is selected. Call the method from the SelectedIndexChanged event when item 0 is selected in the first ListBox. When you programme runs, the Cheque payment options should display in your second ListBox, instead of the Credit Card options.