In some early printings of *Head First C#* (4th edition), pages 77 and 78 were reversed! These are the corrected pages, with the steps in the proper order.

Add C# code to update the TextBlock

In Chapter 1 you added **event handlers**—methods that are called when a certain event is **raised** (sometimes we say the event is **triggered** or **fired**)—to handle mouse clicks in your animal matching game. Now we'll add an event handler to the code-behind that's called any time the user enters text into the TextBox and copies that text to the TextBlock that you added to the upper-right cell in the mini-exercise.

When you double-click on a TextBox control, the IDE adds an event handler for the TextChanged event that's called any time the user changes its text. Double-clicking on other types of controls might add other event handlers—and in some cases (like with TextBlock) doesn't add any event handlers at all.

1. **Double-click on the TextBox control to add the method.**

As soon as you double-click on the TextBox, the IDE will **automatically add a C# event handler method** hooked up to its TextChanged event. It generates an empty method and gives it a name that consists of the name of the control (`numberTextBox`) followed by an underscore and the name of the event being handled—numberTextBox_TextChanged:

```
private void numberTextBox_TextChanged(object sender, TextChangedEventArgs e)
{

}
```

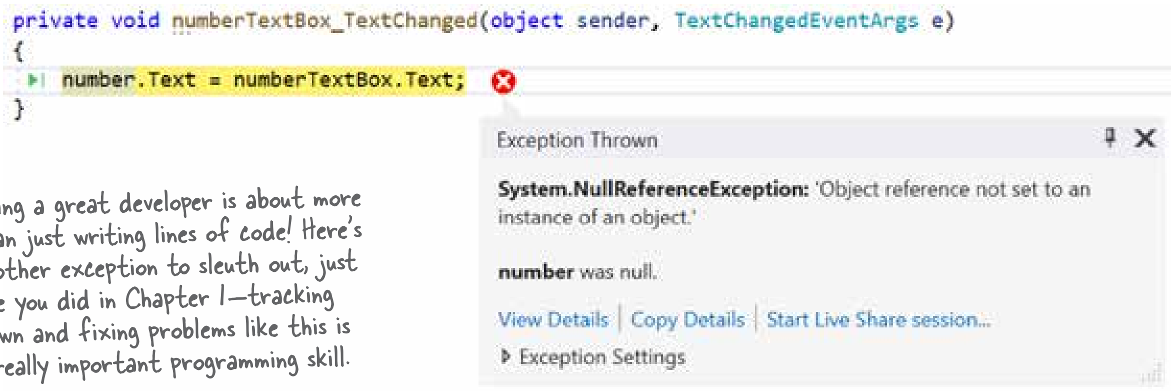
2. **Add code to the new TextChanged event handler.**

Any time the user enters text into the TextBox, we want the app to copy it into the TextBlock that you added to the upper-right cell of the grid. Since you gave the TextBlock a name (`number`) and you also gave the TextBox a name (`numberTextBox`), you just need one line of code to copy its contents:

```
private void numberTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    number.Text = numberTextBox.Text;
}
```

This line of code sets the text in the TextBlock so it's the same as the text in the TextBox, and it gets called any time the user changes the text in the TextBox.

Now run your app. ***Oops! Something went wrong—it threw an exception.***



```
private void numberTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    number.Text = numberTextBox.Text;
}
```

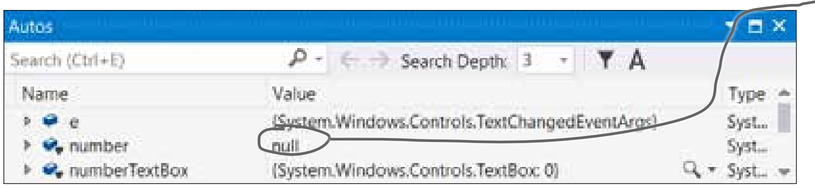
Exception Thrown

System.NullReferenceException: 'Object reference not set to an instance of an object.'

number was null.

Being a great developer is about more than just writing lines of code! Here's another exception to sleuth out, just like you did in Chapter 1—tracking down and fixing problems like this is a really important programming skill.

Take a look at the bottom of the IDE. It has an Autos window that shows you any defined variables. If you don't see, choose *Debug* >> *Windows* >> *Autos* from the menu.



Name	Value	Type
e	{System.Windows.Controls.TextChangedEventArgs}	Syst...
number	null	Syst...
numberTextBox	{System.Windows.Controls.TextBox: 0}	Syst...

The number TextBox says "null"—and we see that same word in NullReferenceException.

So what's going on—***and, more importantly, how do we fix it?***

Sleuth it out

The Autos window is showing you the variables used by the statement that threw the exception: `number` and `numberTextBox`. The value of `numberTextBox` is *{System.Windows.Controls.TextBox: 0}*, and that's what a healthy TextBox looks like in the debugger. But the value of `number`—the TextBlock that you're trying to copy the text to—is **null**. You'll learn more about what null means later in the book.

But here's the all-important clue: the IDE is telling you that the **`number` TextBlock is not initialized**.

The problem is that the XAML for the TextBox includes `Text="0"`, so when the app starts running it initializes the TextBox and tries to set the text. That fires the TextChanged event handler, which tries to copy the text to the TextBlock. But the TextBlock is still null, so the app throws an exception.

So all we need to do to fix the bug is to make sure the TextBlock is initialized before the TextBox. When a WPF app starts up, the controls are **initialized in the order they appear in the XAML**. So you can fix the bug by ***changing the order of the controls in the XAML***.

Swap the order of the TextBlock and TextBox controls so the TextBlock appears above the TextBox:

Moving the TextBlock tag in the XAML so it's above the TextBox causes the TextBlock to get initialized first.

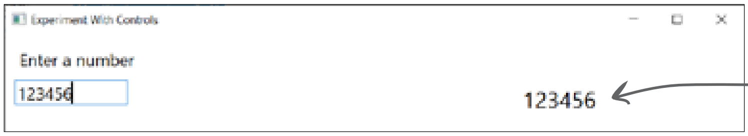
```
<Label Content="Enter a number" ... />
<TextBlock x:Name="number" Grid.Column="1" ... />
<TextBox x:Name="numberTextBox" ... />
```

Select the TextBlock tag in the XAML editor move it above the TextBox so it gets initialized first.

The app should still look exactly the same in the designer—which makes sense, because it still has the same controls. Now run your app again. This time it starts up, and the TextBox now only accepts numeric input.

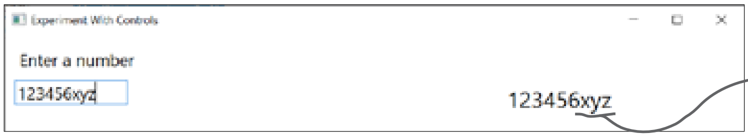
3 **Run your app and try out the TextBox.**

Use the Start Debugging button (or choose Start Debugging (F5) from the Debug menu) to start your app, just like you did with the animal matching game in Chapter 1. (If the runtime tools appear, you can disable them just like you did in Chapter 1.) Type any number into the TextBox and it will get copied.



When you type a number into the TextBox, the TextChange event handler copies it to the TextBlock.

But something's wrong—you can enter any text into the TextBox, not just numbers!



There has to be a way to allow the user to enter only numbers! How do you think we'll do that?