A developer typically creates a class to represent an entity (table) in the database, whereby CRUD operations are performed by calling methods on the objects, e.g; for class Employee, objEmployee.Save(), objEmployee.Load(), objEmployee.Update(), objEmployee.Delete() etc. Also, everytime the values in the object change, the developer has to manually write code to update each of the UI elements that display the values stored in the object's members. The same holds true when the values on the UI change. With WPF, you can lessen your code greatly by binding the object directly with the UI (each of object's members data-bound to individual UI elements) using XAML syntax. I will be covering 3 different approaches for displaying data in the object onto the UI.

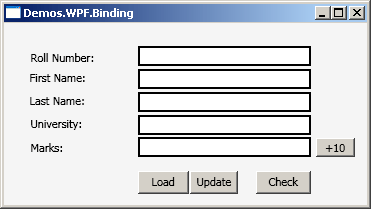
1. Typical
2. Using only C# [INotifyPropertyChanged]
3. Using C# and XAML [INotifyPropertyChanged and XAML]

I am going to use the "Examinee" table. The columns for the table are

* RollNo (PK, nvarchar(5), not null)
* FirstName (nvarchar(10), not null)
* LastName (nvarchar(10), not null)
* University (nvarchar(50), not null)
* MarksObtained (int, not null)
* Exam (nvarchar(15), not null)

**Typical Approach**  
The following provides a glimpse of what goes on in code with the typical approach. The developer creates the Examinee class for the table as follows: [For clarity, code has been colored and lines numbered. Also, ignore the SqlDataReader use for fetching a single row; it was done to lessen code ;)]

// =========================  
 // Code Listing 1  
 // =========================  
 using System;  
 using System.Collections.Generic;  
 using System.Text;  
 using System.Data;  
 using System.Data.SqlClient;  
   
 namespace Demos.WPF.Binding.PropertyChangeNotNotified  
 {  
 class Examinee  
 {  
 private string \_RollNo;  
 public string RollNo  
 {  
 14: get { return \_RollNo; }  
 15: set { \_RollNo = value; }  
 16: }  
 17:   
 18: private string \_FirstName;  
 19: public string FirstName  
 20: {  
 21: get { return \_FirstName; }  
 22: set { \_FirstName = value; }  
 23: }  
 24:   
 25: private string \_LastName;  
 26: public string LastName  
 27: {  
 28: get { return \_LastName; }  
 29: set { \_LastName = value; }  
 30: }  
 31:   
 32: private string \_University;  
 33: public string University  
 34: {  
 35: get { return \_University; }  
 36: set { \_University = value; }  
 37: }  
 38:   
 39: private int \_MarksObtained;  
 40: public int MarksObtained  
 41: {  
 42: get { return \_MarksObtained; }  
 43: set { \_MarksObtained = value; }  
 44: }  
 45:   
 46: private string \_Exam;  
 47: public string Exam  
 48: {  
 49: get { return \_Exam; }  
 50: set { \_Exam = value; }  
 51: }  
 52:   
 53: public Examinee()  
 54: {  
 55: }  
 56:   
 57: public void Load()  
 58: {  
 59: if (\_RollNo.Trim().Length == 0)  
 60: throw new Exception("Roll Number not specified.");  
 61:   
 62: SqlConnection oConnection = new SqlConnection([...]);  
 63: oConnection.Open();  
 64:   
 65: string str = "";  
 66: str += "SELECT \* ";  
 67: str += "FROM Examinee ";  
 68: str += "WHERE RollNo = @RollNo";  
 69: SqlCommand oCommand = new SqlCommand(str, oConnection);  
 70: oCommand.CommandType = CommandType.Text;  
 71: oCommand.Parameters.Add(new SqlParameter("@RollNo", \_RollNo));  
 72: SqlDataReader oDR = oCommand.ExecuteReader();  
 73:   
 74: if (oDR.Read())  
 75: {  
 76: \_FirstName = oDR["FirstName"].ToString();  
 77: \_LastName = oDR["LastName"].ToString();  
 78: \_University = oDR["University"].ToString();  
 79: \_MarksObtained = Convert.ToInt32(oDR["MarksObtained"]);  
 80: \_Exam = oDR["Exam"].ToString();  
 81: }  
 82:   
 83: oCommand.Dispose();  
 84: oConnection.Close();  
 85: oConnection.Dispose();  
 86: }  
 87:   
 88: public void Update()  
 89: {  
 90: if (\_RollNo.Trim().Length == 0)  
 91: throw new Exception("Roll Number not specified.");  
 92:   
 93: SqlConnection oConnection = new SqlConnection("[...]");  
 94: oConnection.Open();  
 95:   
 96: string str = "";  
 97: str += "UPDATE Examinee ";  
 98: str += "SET FirstName = @FirstName, ";  
 99: str += "LastName = @LastName, ";  
100: str += "University = @University, ";  
101: str += "MarksObtained = @Marks ";  
102: SqlCommand oCommand = new SqlCommand(str, oConnection);  
103: oCommand.CommandType = CommandType.Text;  
104: oCommand.Parameters.Add(new SqlParameter("@FirstName", \_FirstName));  
105: oCommand.Parameters.Add(new SqlParameter("@LastName", \_LastName));  
106: oCommand.Parameters.Add(new SqlParameter("@University", \_University));  
107: oCommand.Parameters.Add(new SqlParameter("@Marks", \_MarksObtained));  
108: oCommand.Parameters.Add(new SqlParameter("@RollNo", \_RollNo));  
109: oCommand.ExecuteNonQuery();  
110:   
111: oCommand.Dispose();  
112: oConnection.Close();  
113: oConnection.Dispose();  
114: }  
115: }  
116: }

I figured that since it is XAML that we are demonstrating, might as well develop the UI in Microsoft Cider, the Visual Studio 2005 add-in visual designer for XAML. So, to use the above class, the UI would be something like this.  
  
  
  
The resulting XAML for the window shown above is

<!--   
 =========================  
 Code Listing 2  
 Simple.xaml  
 =========================  
 -->  
<Window   
 x:Class="Demos.WPF.Binding.Simple"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 Title="Demos.WPF.Binding" Height="209" Width="371"  
 Background="WhiteSmoke" WindowStartupLocation="CenterScreen">  
 <Grid>  
 <Label  
 VerticalAlignment="Top"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="21.37,23,0,0"   
 Width="76.63"   
 Height="23.2766666666667"   
 Name="label1">Roll Number:</Label>  
 <Label   
 VerticalAlignment="Top"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="20.37,42.7233333333333,0,0"   
 Width="76.63"   
 Height="23.2766666666667"   
 Name="label2">First Name:</Label>  
 <Label   
 VerticalAlignment="Top"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="20.37,66,0,0"   
 Width="75.63"   
 Height="23.2766666666667"   
 Name="label3">Last Name:</Label>  
 <Label   
 VerticalAlignment="Top"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="21.37,89,0,0"   
 Width="62.63"   
 Height="23.2766666666667"   
 Name="label4">University:</Label>  
 <Label   
 VerticalAlignment="Top"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="21.37,112.723333333333,0,0"   
 Width="62.63"   
 Height="23.276666666666671"   
 Name="label5">Marks:</Label>  
 <TextBox   
 VerticalAlignment="Top"   
 HorizontalAlignment="Stretch"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,23,56,0"   
 Width="NaN"   
 Height="20"   
 Name="txtRollNo"   
 BorderBrush="#FF000000"   
 Foreground="#FF336699"></TextBox>  
 <TextBox   
 VerticalAlignment="Top"   
 HorizontalAlignment="Stretch"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,46,56,0"   
 Width="NaN"   
 Height="20"   
 Name="txtFirstName"   
 Foreground="#FF336699"   
 BorderBrush="#FF000000"></TextBox>  
 <TextBox   
 VerticalAlignment="Top"   
 HorizontalAlignment="Stretch"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,69.2766666666667,56,0"   
 Width="NaN"   
 Height="20"   
 Name="txtLastName"   
 Foreground="#FF336699"   
 BorderBrush="#FF000000"></TextBox>  
 <TextBox   
 VerticalAlignment="Top"   
 HorizontalAlignment="Stretch"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,92.2766666666667,56,0"   
 Width="NaN"   
 Height="20"   
 Name="txtUniversity"   
 Foreground="#FF336699"   
 BorderBrush="#FF000000"></TextBox>  
 <TextBox   
 VerticalAlignment="Top"   
 HorizontalAlignment="Stretch"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,114,56,0"   
 Width="NaN"   
 Height="20"   
 Name="txtMarks"   
 Foreground="#FF336699"   
 BorderBrush="#FF000000"></TextBox>  
 <Button   
 VerticalAlignment="Bottom"   
 HorizontalAlignment="Left"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="134,0,0,11"   
 Width="51"   
 Height="23"   
 Name="btnLoad"   
 Click="btnLoad\_Click">Load</Button>  
 <Button   
 VerticalAlignment="Bottom"   
 HorizontalAlignment="Right"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="0,0,128.5625,11"   
 Width="48.4375"   
 Height="23"   
 Name="btnUpdate"   
 Click="btnUpdate\_Click">Update</Button>  
 <Button   
 VerticalAlignment="Bottom"   
 HorizontalAlignment="Right"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="0,0,56,11"   
 Width="55"   
 Height="23"   
 Name="btnCheck"   
 Click="btnCheck\_Click">Check</Button>  
 <Button   
 VerticalAlignment="Top"   
 HorizontalAlignment="Right"   
 Grid.Column="0"   
 Grid.ColumnSpan="1"   
 Grid.Row="0"   
 Grid.RowSpan="1"   
 Margin="0,115,12,0"   
 Width="39"   
 Height="19"   
 Name="btnAddTen"   
 Click="btnAddTen\_Click">+10</Button>  
 </Grid>  
</Window>

The app lets the user enter the Roll Number for a student, and dislays the Examinee's stats when the "Load" button is clicked. The user can then make any changes in values if required (except for the Roll Number), and click "Update" to make the changes in the database. Nothing fancy here. The code-behind file is pretty straight-forward as well. I have placed a "Check" button on the form to let the user view and compare the values in both, the Examinee object and on the UI.

// =========================  
 // Code Listing 3  
 // Simple.xaml.cs  
 // =========================  
 1: using System;  
 2: using System.Windows;  
 3: using System.Windows.Controls;  
 4: using System.Windows.Data;  
 5: using System.Windows.Documents;  
 6: using System.Windows.Media;  
 7: using System.Windows.Media.Imaging;  
 8: using System.Windows.Shapes;  
 9: using Demos.WPF.Binding.PropertyChangeNotNotified;  
 10:   
 11: namespace Demos.WPF.Binding  
 12: {  
 13: public partial class Simple : Window  
 14: {  
 15: Examinee oExaminee = new Examinee();  
 16:   
 17: public Simple()  
 18: {  
 19: InitializeComponent();  
 20: }  
 21:   
 22: private void btnLoad\_Click(object sender, EventArgs e)  
 23: {  
 24: try  
 25: {  
 26: oExaminee.RollNo = txtRollNo.Text;  
 27: oExaminee.Load();  
 28:   
 29: // tedious code to update UI with object's properties  
 30: txtFirstName.Text = oExaminee.FirstName;  
 31: txtLastName.Text = oExaminee.LastName;  
 32: txtUniversity.Text = oExaminee.University;  
 33: txtMarks.Text = oExaminee.MarksObtained.ToString();  
 34: }  
 35: catch (Exception oEx)  
 36: {  
 37: MessageBox.Show(oEx.Message);  
 38: }  
 39: }  
 40:   
 41: private void btnUpdate\_Click(object sender, EventArgs e)  
 42: {  
 43: try  
 44: {  
 45: // tedious code to set object's properties from UI elements  
 46: oExaminee.FirstName = txtFirstName.Text;  
 47: oExaminee.LastName = txtLastName.Text;  
 48: oExaminee.University = txtUniversity.Text;  
 49: oExaminee.MarksObtained = Convert.ToInt32(txtMarks.Text);  
 50:   
 51: oExaminee.Update();  
 52: MessageBox.Show("Information Updated.", "Update");  
 53: }  
 54: catch (Exception oEx)  
 55: {  
 56: MessageBox.Show(oEx.Message);  
 57: }  
 58: }  
 59:   
 60: private void btnCheck\_Click(object sender, EventArgs e)  
 61: {  
 62: string str = string.Format("OBJECT:\nFirstName: {0}"   
 63: + "\nLastName: {1}"  
 64: + "\nUniversity: {2}"  
 65: + "\nMarksObtained: {3}"  
 66: + "\n\nUI:"  
 67: + "\nFirst Name: {4}"  
 68: + "\nLast Name: {5}"  
 69: + "\nUniversity: {6}"  
 70: + "\nMarksObtained: {7}",   
 71: oExaminee.FirstName,   
 72: oExaminee.LastName,   
 73: oExaminee.University,   
 74: oExaminee.MarksObtained.ToString(),   
 75: txtFirstName.Text,   
 76: txtLastName.Text,   
 77: txtUniversity.Text,   
 78: txtMarks.Text);  
 79: MessageBox.Show(str);  
 80: }  
 81:   
 82: private void btnAddTen\_Click(object sender, EventArgs e)  
 83: {  
 84: if ((oExaminee.MarksObtained + 10) > 100)  
 85: oExaminee.MarksObtained = 100;  
 86: else   
 87: oExaminee.MarksObtained += 10;  
 88: }  
 89:   
 90: }  
 91: }

Notice that in the event handlers for both, the "Load" (lines 29 to 33) and "Update" (lines 45 to 49) buttons, the values need to be shuttled between the object and the UI elements to keep them synchronized. If the form had, lets say, 30+ UI elements, the developer would have to write atleast 60 additional lines of code to achieve this synchronization. Ironically, this is what usually happens.  
  
**Using only C# [INotifyPropertyChanged]**  
The typical approach is pretty cumbersome. The same result can be achieved by implementing the INotifyPropertyChanged interface in the Examinee class to generate an event everytime the value in the object's property changes. I have highlighted portions in code listing 4 that need to be added to implement INotifyPropertyChanged.

// =========================  
 // Code Listing 4  
 // =========================  
 1: using System;  
 2: using System.Collections.Generic;  
 3: using System.Text;  
 4: using System.Data;  
 5: using System.Data.SqlClient;  
 6: using System.ComponentModel;  
 7:   
 8: namespace Demos.WPF.Binding.PropertyChangeNotified  
 9: {  
 10: class Examinee : INotifyPropertyChanged  
 11: {  
 12: private string \_RollNo;  
 13: public string RollNo  
 14: {  
 15: get { return \_RollNo; }  
 16: set  
 17: {   
 18: \_RollNo = value;  
 19: // Notify Property Change  
 20: OnPropertyChanged("RollNo");  
 21: }  
 22: }  
 23:   
 24: private string \_FirstName;  
 25: public string FirstName  
 26: {  
 27: get { return \_FirstName; }  
 28: set   
 29: {   
 30: \_FirstName = value;   
 31: // Notify Property Change  
 32: OnPropertyChanged("FirstName");   
 33: }  
 34: }  
 35:   
 36: private string \_LastName;  
 37: public string LastName  
 38: {  
 39: get { return \_LastName; }  
 40: set   
 41: {   
 42: \_LastName = value;   
 43: // Notify Property Change  
 44: OnPropertyChanged("LastName");  
 45: }  
 46: }  
 47:   
 48: private string \_University;  
 49: public string University  
 50: {  
 51: get { return \_University; }  
 52: set   
 53: {   
 54: \_University = value;   
 55: // Notify Property Change  
 56: OnPropertyChanged("University");   
 57: }  
 58: }  
 59:   
 60: private int \_MarksObtained;  
 61: public int MarksObtained  
 62: {  
 63: get { return \_MarksObtained; }  
 64: set   
 65: {   
 66: \_MarksObtained = value;  
 67: // Notify Property Change  
 68: OnPropertyChanged("MarksObtained");   
 69: }  
 70: }  
 71:   
 72: private string \_Exam;  
 73: public string Exam  
 74: {  
 75: get { return \_Exam; }  
 76: set   
 77: {   
 78: \_Exam = value;   
 79: // Notify Property Change  
 80: OnPropertyChanged("Exam");   
 81: }  
 82: }  
 83:   
 84: public Examinee()  
 85: {  
 86: }  
 87:   
 88: public void Load()  
 89: {  
 90: if (\_RollNo.Trim().Length == 0)  
 91: throw new Exception("Roll Number not specified.");  
 92:   
 93: SqlConnection oConnection = new SqlConnection([...]);  
 94: oConnection.Open();  
 95:   
 96: string str = "";  
 97: str += "SELECT \* ";  
 98: str += "FROM Examinee ";  
 99: str += "WHERE RollNo = @RollNo";  
100: SqlCommand oCommand = new SqlCommand(str, oConnection);  
101: oCommand.CommandType = CommandType.Text;  
102: oCommand.Parameters.Add(new SqlParameter("@RollNo", \_RollNo));  
103: SqlDataReader oDR = oCommand.ExecuteReader();  
104:   
105: if (oDR.Read())  
106: {  
107: \_FirstName = oDR["FirstName"].ToString();  
108: \_LastName = oDR["LastName"].ToString();  
109: \_University = oDR["University"].ToString();  
110: \_MarksObtained = Convert.ToInt32(oDR["MarksObtained"]);  
111: \_Exam = oDR["Exam"].ToString();  
112:   
113: // following code raises onPropertyChanged event because   
114: // properties are being set from within the class itself.  
115: OnPropertyChanged("FirstName");   
116: OnPropertyChanged("LastName");   
117: OnPropertyChanged("University");   
118: OnPropertyChanged("MarksObtained");  
119: OnPropertyChanged("Exam");   
120: }  
121:   
122: oCommand.Dispose();  
123: oConnection.Close();  
124: oConnection.Dispose();  
125: }  
126:   
127: public void Update()  
128: {  
129: if (\_RollNo.Trim().Length == 0)  
130: throw new Exception("Roll Number not specified.");  
131:   
132: SqlConnection oConnection = new SqlConnection("[...]");  
133: oConnection.Open();  
134:   
135: string str = "";  
136: str += "UPDATE Examinee ";  
137: str += "SET FirstName = @FirstName, ";  
138: str += "LastName = @LastName, ";  
139: str += "University = @University, ";  
140: str += "MarksObtained = @Marks ";  
141: SqlCommand oCommand = new SqlCommand(str, oConnection);  
142: oCommand.CommandType = CommandType.Text;  
143: oCommand.Parameters.Add(new SqlParameter("@FirstName", \_FirstName));  
144: oCommand.Parameters.Add(new SqlParameter("@LastName", \_LastName));  
145: oCommand.Parameters.Add(new SqlParameter("@University", \_University));  
146: oCommand.Parameters.Add(new SqlParameter("@Marks", \_MarksObtained));  
147: oCommand.Parameters.Add(new SqlParameter("@RollNo", \_RollNo));  
148: oCommand.ExecuteNonQuery();  
149:   
150: oCommand.Dispose();  
151: oConnection.Close();  
152: oConnection.Dispose();  
153: }  
154:   
155: public event PropertyChangedEventHandler PropertyChanged;   
156: protected void OnPropertyChanged(string propertyName)   
157: {   
158: if (this.PropertyChanged != null)   
159: PropertyChanged(this, new PropertyChangedEventArgs(propertyName));  
160: }   
161: }  
162: }

Changes made to class Examinee of listing 1 are given in code listing 4 above. The INotifyPropertyChanged interface implementation allows the class to raise the PropertyChanged event everytime a property value changes. The above class can now be instanitated in the code-behind file (see code listing 5 below) of another XAML file (WithoutXAML.xaml) which is similar to that of code listing 2.

// =========================  
 // Code Listing 5  
 // WithoutXAML.xaml.cs  
 // =========================  
 1: using System;  
 2: using System.Windows;  
 3: using System.Windows.Controls;  
 4: using System.Windows.Data;  
 5: using System.Windows.Documents;  
 6: using System.Windows.Media;  
 7: using System.Windows.Media.Imaging;  
 8: using System.Windows.Shapes;  
 9: using Demos.WPF.Binding.PropertyChangeNotified;  
 10:   
 11: namespace Demos.WPF.Binding  
 12: {  
 13: public partial class WithoutXAML : Window  
 14: {  
 15: Examinee oExaminee = new Examinee();  
 16:   
 17: public Simple()  
 18: {  
 19: InitializeComponent();  
 20: }  
 21:   
 22: private void WithoutXAML\_Load(object sender, EventArgs e)   
 23: {   
 24: oExaminee.PropertyChanged += oExaminee\_PropertyChanged;  
 25: }   
 26:  
 27: // Property Changed Event Handler [Pathetic Code!]  
 28: private void oExaminee\_PropertyChanged(object sender,   
 29: PropertyChangedEventArgs e)   
 30: {   
 31: switch(e.PropertyName)   
 32: {   
 33: case "FirstName":   
 34: txtFirstName.Text = oExaminee.FirstName;   
 35: break;   
 36: case "LastName":   
 37: txtLastName.Text = oExaminee.LastName;   
 38: break;   
 39: case "University":   
 40: txtUniversity.Text = oExaminee.University;   
 41: break;   
 42: case "MarksObtained":   
 43: txtMarks.Text = oExaminee.MarksObtained.ToString();  
 44: break;   
 45: }   
 46: }   
 47:   
 48: private void btnLoad\_Click(object sender, EventArgs e)  
 49: {  
 50: try  
 51: {  
 52: oExaminee.RollNo = txtRollNo.Text;  
 53: oExaminee.Load();  
 54: }  
 55: catch (Exception oEx)  
 56: {  
 57: MessageBox.Show(oEx.Message);  
 58: }  
 59: }  
 60:   
 61: private void btnUpdate\_Click(object sender, EventArgs e)  
 62: {  
 63: try  
 64: {  
 65: oExaminee.Update();  
 66: MessageBox.Show("Information Updated.", "Update");  
 67: }  
 68: catch (Exception oEx)  
 69: {  
 70: MessageBox.Show(oEx.Message);  
 71: }  
 72: }  
 73:   
 74: private void btnCheck\_Click(object sender, EventArgs e)  
 75: {  
 76: string str = string.Format("OBJECT:\nFirstName: {0}"   
 77: + "\nLastName: {1}"  
 78: + "\nUniversity: {2}"  
 79: + "\nMarksObtained: {3}"  
 80: + "\n\nUI:"  
 81: + "\nFirst Name: {4}"  
 82: + "\nLast Name: {5}"  
 83: + "\nUniversity: {6}"  
 84: + "\nMarksObtained: {7}",   
 85: oExaminee.FirstName,   
 86: oExaminee.LastName,   
 87: oExaminee.University,   
 88: oExaminee.MarksObtained.ToString(),   
 89: txtFirstName.Text,   
 90: txtLastName.Text,   
 91: txtUniversity.Text,   
 92: txtMarks.Text);  
 93: MessageBox.Show(str);  
 94: }  
 95:   
 96: private void btnAddTen\_Click(object sender, EventArgs e)  
 97: {  
 98: if ((oExaminee.MarksObtained + 10) > 100)  
 99: oExaminee.MarksObtained = 100;  
100: else   
101: oExaminee.MarksObtained += 10;  
102: }  
103:   
104: ///////////////////////////////////  
105: // WARNING: Pathetic Code below! //  
106: // Event-handlers for text-boxes //  
107: ///////////////////////////////////  
108:   
109: private void txtRollNo\_TextChanged(object sender,   
110: EventArgs e)   
111: {   
112: oExaminee.RollNo = txtRollNo.Text;   
113: }   
114:   
115: private void txtFirstName\_TextChanged(object sender,   
116: EventArgs e)   
117: {   
118: oExaminee.FirstName = txtFirstName.Text;   
119: }   
120:   
121: private void txtLastName\_TextChanged(object sender,   
122: EventArgs e)   
123: {   
124: oExaminee.LastName = txtLastName.Text;   
125: }   
126:   
127: private void txtUniversity\_TextChanged(object sender,   
128: EventArgs e)   
129: {   
130: oExaminee.University = txtUniversity.Text;   
131: }   
132:   
133: private void txtMarks\_TextChanged(object sender,   
134: EventArgs e)   
135: {   
136: int marks = 0;   
137: if (int.TryParse(txtMarks.Text, out marks))   
138: oExaminee.MarksObtained = marks;   
139: }   
140: }  
141: }

Notice that the oExaminee\_PropertyChanged event handler (lines 27 to 46) updates the UI with object values everytime a PropertyChanged event occurs on the Examinee object. To update the object with UI values, TextChanged event handlers were added for all the textboxes (see lines 104 to 139).  
  
Notice also that the "Load" and "Update" button event handlers do not contain the Object-to-UI and UI-to-Object update code as that is now being handles by the event handlers.  
  
However, just because we implemented the INotifyPropertyChanged interface in our Examinee class, we were not absolved of the responsibility to write custom event handlers for the object and UI update logic. The whole point of all the above code is to show the tedious coding required to keep the object and UI in sync.  
  
**Using C# and XAML [INotifyPropertyChanged and XAML]**  
Both the approaches described above were used with typical Windows applications, although I created the form using XAML. The final approach (and the right one at that) uses the Data Binding syntax in XAML, and would show you how easy it is to bind UI elements directly to an object that implements INotifyPropertyChanged.  
  
To see WPF Data Binding in action, the Binding sytax in the textboxes' "Text" property values. The XAML file would not look something like code listing 6. [Changes from previous XAML file have been highlighted.]

<!--   
 =========================  
 Code Listing 6  
 WithXAML.xaml  
 =========================  
 -->  
<Window   
 x:Class="Demos.WPF.Binding.WithXAML"  
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
 Title="Binding - Using XAML Binding Syntax"   
 Height="209" Width="371" Background="WhiteSmoke"  
 WindowStartupLocation="CenterScreen" Loaded="WithXAML\_Load" >  
 <Grid x:Name="myGrid" >  
 <Label ...>Roll Number:</Label>  
 <Label ...>First Name:</Label>  
 <Label ...>Last Name:</Label>  
 <Label ...>University:</Label>  
 <Label ...>Marks:</Label>  
 <TextBox  
 ...  
 Name="txtRollNo"   
 Text="{Binding Path=RollNo}"></TextBox>  
 <TextBox   
 ...  
 Name="txtFirstName"   
 Text="{Binding Path=FirstName}"></TextBox>  
 <TextBox  
 ...  
 Name="txtLastName"   
 Text="{Binding Path=LastName}"></TextBox>  
 <TextBox  
 ...  
 Name="txtUniversity"   
 Text="{Binding Path=University}"></TextBox>  
 <TextBox  
 ...   
 Name="txtMarks"   
 Text="{Binding Path=MarksObtained}"></TextBox>  
 <Button ...>Load</Button>  
 <Button ...>Update</Button>  
 <Button ...>Check</Button>  
 <Button ...>+10</Button>  
 </Grid>  
</Window>

The Binding statement added to the XAML above allows you to discard all the event-handlers in the code-behind that were needed before. The function that we wanted from our app is now achievable with a lot less code.

// =========================  
 // Code Listing 7  
 // WithXAML.xaml.cs  
 // =========================  
 1: using System;  
 2: using System.Windows;  
 3: using System.Windows.Controls;  
 4: using System.Windows.Data;  
 5: using System.Windows.Documents;  
 6: using System.Windows.Media;  
 7: using System.Windows.Media.Imaging;  
 8: using System.Windows.Shapes;  
 9: using Demos.WPF.Binding.PropertyChangeNotified;  
 10:   
 11: namespace Demos.WPF.Binding  
 12: {  
 13: public partial class WithXAML : Window  
 14: {  
 15: Examinee oExaminee = new Examinee();  
 16:   
 17: public WithXAML()  
 18: {  
 19: InitializeComponent();  
 20: }  
 21:   
 22: private void WithXAML\_Load(object sender, EventArgs e)  
 23: {  
 24: myGrid.DataContext = oExaminee;  
 25: }  
 26:   
 27: private void btnAddTen\_Click(object sender, EventArgs e)  
 28: {  
 29: if ((oExaminee.MarksObtained + 10) > 100)  
 30: oExaminee.MarksObtained = 100;  
 31: else  
 32: oExaminee.MarksObtained += 10;  
 33: }  
 34:   
 35: private void btnLoad\_Click(object sender, EventArgs e)  
 36: {  
 37: try  
 38: {  
 39: oExaminee.Load();  
 40: }  
 41: catch (Exception oEx)  
 42: {  
 43: MessageBox.Show(oEx.Message);  
 44: }  
 45: }  
 46:   
 47: private void btnUpdate\_Click(object sender, EventArgs e)  
 48: {  
 49: try  
 50: {  
 51: oExaminee.Update();  
 52: MessageBox.Show("Information Updated.", "Update");  
 53: }  
 54: catch (Exception oEx)  
 55: {  
 56: MessageBox.Show(oEx.Message);  
 57: }  
 58: }  
 59:   
 60: private void btnCheck\_Click(object sender, EventArgs e)  
 61: {  
 62: string str = string.Format("OBJECT:\nFirstName: {0}"  
 63: + "\nLastName: {1}"  
 64: + "\nUniversity: {2}"  
 65: + "\nMarksObtained: {3}"  
 66: + "\n\nUI:"  
 67: + "\nFirst Name: {4}"  
 68: + "\nLast Name: {5}"  
 69: + "\nUniversity: {6}"  
 70: + "\nMarksObtained: {7}",  
 71: oExaminee.FirstName,  
 72: oExaminee.LastName,  
 73: oExaminee.University,  
 74: oExaminee.MarksObtained.ToString(),  
 75: txtFirstName.Text,  
 76: txtLastName.Text,  
 77: txtUniversity.Text,  
 78: txtMarks.Text);  
 79: MessageBox.Show(str);  
 80: }  
 81: }  
 82: }

The only change occured in line 24 in the WithXAML\_Load event handler; the Examinee object was assigned to the main Grid's DataContext property. The highlighted statement in code listing 7 above defines a data context for the Grid, the UI element that encloses all the other UI controls. WPF introduces the concept of "Dependency Properties" that enables a XAML UI element to inherit a value from its parent UI control (in this case the grid). Once the data context is set in the form load event, the {Binding Path=...} statements in XAML simply retrieve the values from the bound object or resource and place them in the target textbox. If you run the application now, everytime you change the value in a textbox, the corresponding value in the object changes (You can test this by clicking the "Check" button after you update the value in a textbox). Same holds true for the object. If you click the "+10" (btnAddTen) button, the value in the object is incremented by 10. However, you will also see that the value is automatically updated in the textbox as well.