LAB MANUAL: CSE 3200

SOFTWARE DEVELOPMENT V

CONTENTS

LAB 01: INTRODUCTION TO .NET AND VISUAL C#

DOWNLOADING VISUAL STUDIO

Download and install Visual Studio 2010 or upper Full Version and choose the **Visual C#** template/environment when creating a new project.

You can find different versions of Visual Studio at this address: http://www.visualstudio.com/downloads/download-visual-studio-vs

CREATING A SIMPLE CONSOLE APPLICATION WHICH DISPLAYS HELLO WORLD

1. Create a new project and select **Console Application**
2. Give a **project name**
3. The **namespace** keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally unique types.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Output:

```
Hello World!
```

TAKING NON-NUMERIC DATA FROM KEYBOARD INTO CONSOLE APPLICATION

1. ReadLine () function works similar to scanf () function. **Waits for the user until an input is given from the keyboard**
2. Write () and WriteLine () functions work similar to printf () function

```
namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            string Name = "";
            Console.Write("Please Enter your name: ");
            Name = Console.ReadLine();
            Console.WriteLine ("User Name: " + Name);
        }
    }
}
```

```
Please Enter your name: Mamun
User Name: Mamun
```

TAKING NUMERIC DATA IN CONSOLE APPLICATION

1. ReadLine () function returns a 'string', so in case we want to work with an integer number we have to convert the string to integer by using **Convert.ToInt16 (**string**).** According to the integer's size you can also use **ToInt32, ToInt64** etc.

```
namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            int Age = 0;
            Console.Write("Please Enter your age: ");
            Age = Convert.ToInt16(Console.ReadLine());
            Console.WriteLine("User Age: " + Age);
        }
    }
}
```

```
Please Enter your age: 25
User Age: 25
```

HANDLING ERRORS USING TRY AND CATCH BLOCK

1. Similar to Java, in Visual C# we have to use try and catch block to handle different types of errors.

```
namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            int Age = 0;
            Console.Write("Please Enter your age: ");
            try
            {
                Age = Convert.ToInt16(Console.ReadLine());
                Console.WriteLine("User Name: " + Age);
            }
            catch (Exception)
            {
                Console.WriteLine("You must Enter Numeric value as your age.");
            }
        }
    }
}
```

Sample Output:

Please Enter your age: Mamun

You must Enter Numeric value as your age.

LAB 02: C# CONSOLE APPLICATION

USING "IF" AND "ELSE" TO DEFINE CONDITIONS IN C# APPLICATIONS

1. This program takes a name as input which should be 10 characters long. If not then it will display an error message.

```
namespace Lab02
{
    class Program
    {
        static void Main(string[] args)
        {
            string Name = "";
            Console.Write("Please enter your name: ");
            Name = Console.ReadLine();
            if (Name.Length > 10)
                Console.WriteLine("Name must be within 10 characters.");
            else
                Console.WriteLine("User Name: " + Name);
        }
    }
}
```

```
Please enter your name: Mamunur Rashid Akand
Name must be within 10 characters.
```

PUTTING COMMENTS IN C# CODE

1. Comments in Visual C# are same as C programming language.

```
class Program
{
    static void Main(string[] args)
    {
        string Name = "";
        Console.Write("Please enter your name: "); // You can put a single line comment
        /* or, if you wish,
         * you can put a multiple line comment
         * like this
         * */
    }
}
```

USING FOR LOOP

1. This program prints 0 to 9 using **for** loop.

```
class Program
{
    static void Main(string[] args)
    {
        for (int i = 0; i < 10; i++)
            Console.Write(i);
    }
}
```

```
Sample Output:
0123456789
```

CREATING A SIMPLE CONSOLE CALCULATOR

1. This program takes 2 number as input using ReadLine () function. As we have said earlier, ReadLine () function only takes input as string datatype. So, to calculate real numbers we need to use datatype such as double, float etc.
2. To convert to double we can use the **ToDouble ()** function and to convert to float use **ToSingle ()** function.
3. Then we have to take another input that is the **operator.** Because we have to specify which type of operation we want to perform.
4. If we give operator input anything except +,-,*,/ then we are showing an error message: **Invalid Operator!**

```
namespace Lab02
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter first Number: ");
            double n1 = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter second Number: ");
            double n2 = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter operator: ");
            string op = Console.ReadLine();

            double result = 0;
            bool valid = true;
            switch (op)
            {
                case "+":
                    result = n1 + n2;
                    break;
                case "-":
                    result = n1 - n2;
                    break;
                case "*":
                    result = n1 * n2;
                    break;
                case "/":
                    result = n1 / n2;
                    break;
                default:
                    valid = false;
                    break;
            }
            if (valid) Console.WriteLine("Result is: " + result);
            else Console.WriteLine("Invalid Operator!");
        }
    }
}
```

```
Enter first Number: 4
Enter second Number: 2
Enter operator: +
Result is: 6
```

```
Enter first Number: 4
Enter second Number: 2
Enter operator: 2
Invalid Operator!
```

LAB 03: C# WINDOWS APPLICATION

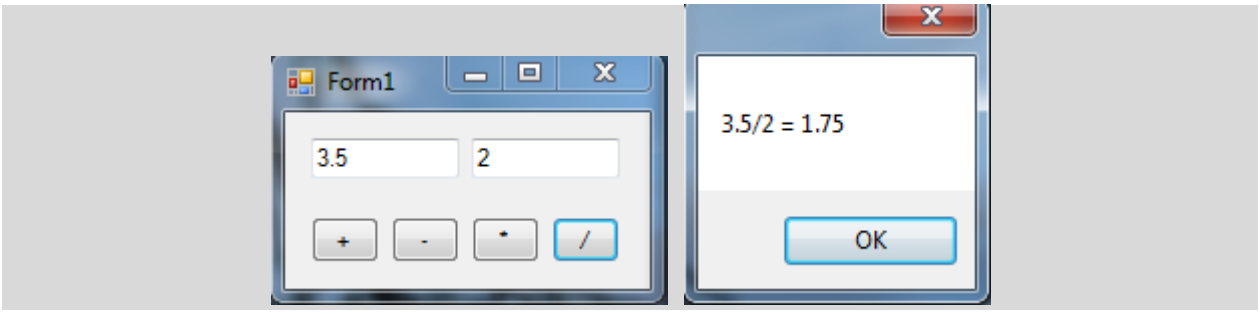
CREATING WINDOWS APPLICATION

1. Until now all we have done is created console applications which have no good representation of Graphical User Interface (GUI).

2. For doing this, we need to create windows application that can have single or multiple forms which are the means of GUI representation.
3. For the program below, we have to first **Create a new project → Select Windows Form Application → Give a proper project name**
4. Then go to **View → ToolBox.** Drag 2 TextBoxes from the ToolBox to the Design form named Form1. In these 2 textboxes we will input 2 numbers for calculation. Give the TextBoxes unique names: **Select the TextBox → Right Click → Properties → In Properties find 'Name' field and give a unique name to the Textbox.** Suppose the 1st TextBox name is **txtNumber1** and the 2nd TextBox name is **txtNumber2.**
5. Then we have to go to **View → ToolBox** and add 4 Buttons for the operations (+,-,*, /). Similarly give each Button a unique name in the **'Name'** field. Here we have given the Button's name as follows: btnPlus, btnMinus, btnMultiplication and btnDivision. We also have to show the symbols (+,-,*, /) on the Button. To do that, **Select the 1st Button →Right Click → Properties → Find 'Text' field and write +. Do this for the other buttons as well.**
6. We want to click the Buttons and perform the operations accordingly. So we have to write some code for each Button's ClickEvent. To do that, double click each button in the form and then write the corresponding operations for the 2 numbers given input in the Textboxes.
7. Here we are showing the output using a **MessageBox.**

```
using System.Windows.Forms;

namespace SimpleForm
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnPlus_Click(object sender, EventArgs e)
        {
            double n1 = Convert.ToDouble(txtNumber1.Text);
            double n2 = Convert.ToDouble(txtNumber2.Text);
            double result = n1 + n2;
            MessageBox.Show(n1 + "+" + n2 + " = " + result);
        }

        private void btnMinus_Click(object sender, EventArgs e)
        {
            double n1 = Convert.ToDouble(txtNumber1.Text);
            double n2 = Convert.ToDouble(txtNumber2.Text);
            double result = n1 - n2;
            MessageBox.Show(n1 + "-" + n2 + " = " + result);
        }

        private void btnMultiplication_Click(object sender, EventArgs e)
```

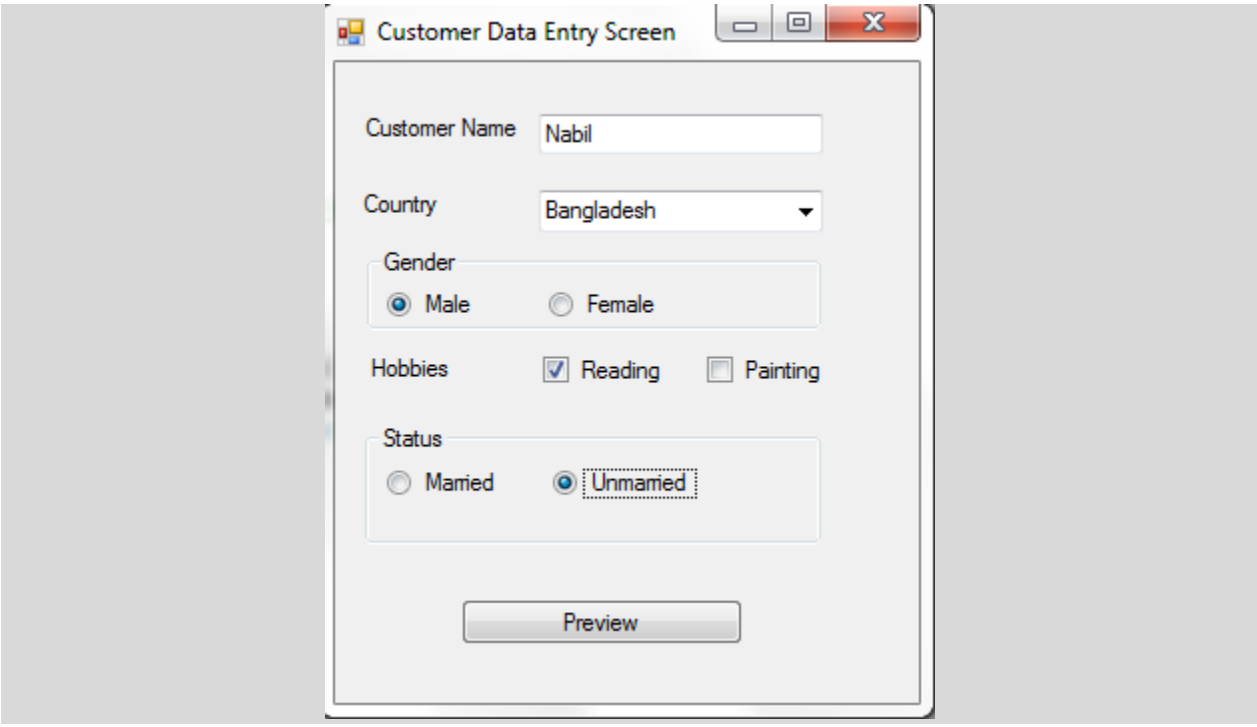
```
        {
            double n1 = Convert.ToDouble(txtNumber1.Text);
            double n2 = Convert.ToDouble(txtNumber2.Text);
            double result = n1 * n2;
            MessageBox.Show(n1 + "*" + n2 + " = " + result);
        }

        private void btnDivision_Click(object sender, EventArgs e)
        {
            double n1 = Convert.ToDouble(txtNumber1.Text);
            double n2 = Convert.ToDouble(txtNumber2.Text);
            double result = n1 / n2;
            MessageBox.Show(n1 + "/" + n2 + " = " + result);
        }
    }
}
```



CREATING A SIMPLE CUSTOMER SCREEN WHICH TAKES CUSTOMER NAME, COUNTRY, GENDER, HOBBY AND STATUS

1. Create a Windows Form and in the 'Text' Properties of the Form write: Customer Data Entry Screen
2. Add Labels from the ToolBox and Add corresponding TextBoxes for Name and Country field.
3. Use radioButton when you want to select a single option from multiple choices.
4. To keep the radioButtons in a group, first Drag a **GroupBox** and inside it drag the radioButtons.
5. Use checkbox when you want to select multiple options from a set of choices.
6. Give unique names for each radioButtons and checkboxes
7. In the following form radioButton for Male is named as **radioMale** and for Female as **radioFemale**
8. Similarly the checkbox for Reading and Painting are named as **chkReading** and **chkPainting**.

9. radioButton for Married is named as **radioMarried** and for Unmarried as **radioUnmarried.**
10. Finally, a **Preview Button** is added at the bottom of the form which when clicked will show the given data in another form. Name the preview button as **btnPreview.**



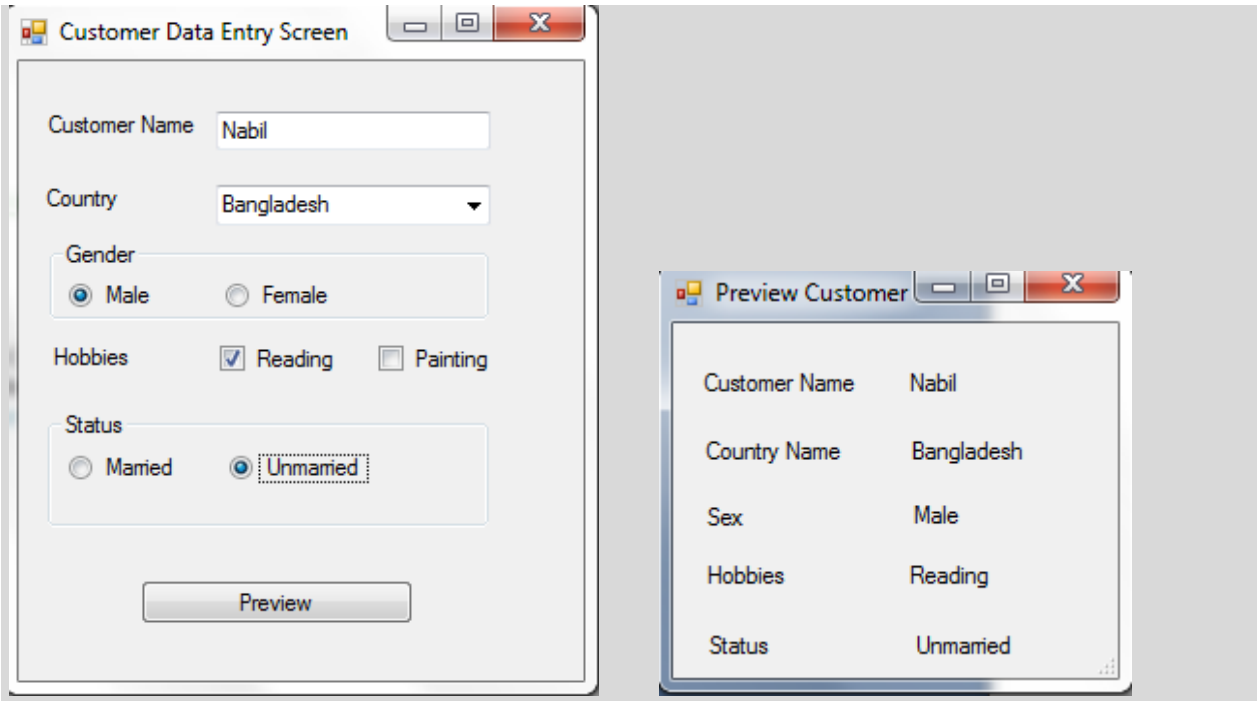
CREATING A PREVIEW SCREEN THAT WILL DISPLAY DATA ENTERED IN TO THE CUSTOMER DATA ENTRY SCREEN

1. In the Customer Data Entry form, double click the button **Preview** and write down the functionality of the clicking event that is show a form which will contain 5 labels for the titles and another 5 Labels to show the data that was given as input.
2. Write a user-defined function SetValues (. . .) that sets the value of the given input to the Labels.

```
namespace CustomerDataEntry
{
    public partial class frmCustomerDataEntry : Form
    {
        public frmCustomerDataEntry()
        {
            InitializeComponent();
        }
```

```
        private void btnPreview_Click(object sender, EventArgs e)
        {
            string Gender, Hobby, Status = "";

            if(radioMale.Checked) Gender = "Male";
            else Gender = "Female";
            if(chkReading.Checked) Hobby = "Reading";
            else Hobby = "Painting";
            if(radioMarried.Checked) Status = "Married";
            else Status = "Unmarried";

            frmCustomerPreview objPreview = new frmCustomerPreview();
            objPreview.SetValues(txtName.Text, cmbCountry.Text, Gender, Hobby, Status);

            objPreview.Show();
        }
    }
}
```

```
namespace CustomerDataEntry
{
    public partial class frmCustomerPreview : Form
    {
        public frmCustomerPreview()
        {
            InitializeComponent();
        }

        public void SetValues(string Name, string Country, string Gender,
                              string Hobby, string Status)
        {
            lblName.Text = Name;
            lblCountry.Text = Country;
            lblGender.Text = Gender;
            lblHobby.Text = Hobby;
            lblStatus.Text = Status;
        }
    }
}
```



LAB 04: NAVIGATIONAL MENUS, OOP AND CONNECTING TO DATABASE

CREATING NAVIGATIONAL MENUS WHICH WILL HELP US TO NAVIGATE CUSTOMER ENTRY SCREEN AND DISPLAY SCREENS EASILY

1. To create a user friendly interface we can use something called MDIParent Form and name it as **MDICustomer.**
2. **View → Solution Explorer → Select your project → Right Click → Add → New Item →** Choose **MDIParent Form** and then add it.
3. In the Design View of MDIParent, Click **File** and you will find some menu options
4. Edit a menu name and give it **Enter Customer**
5. Then double click the **Enter Customer** menu
6. When this menu is clicked you have to invoke the **Customer Data Entry Form** which is named here as **frmCustomerDataEntry**
7. To keep the form within the MDIParent select the **obj.MdiParent = this;**
8. To show the form use the function Show ()
9. In the program.cs file specify the form **MDICustomer.** This will invoke the **MDICustomer** form when the application is run.

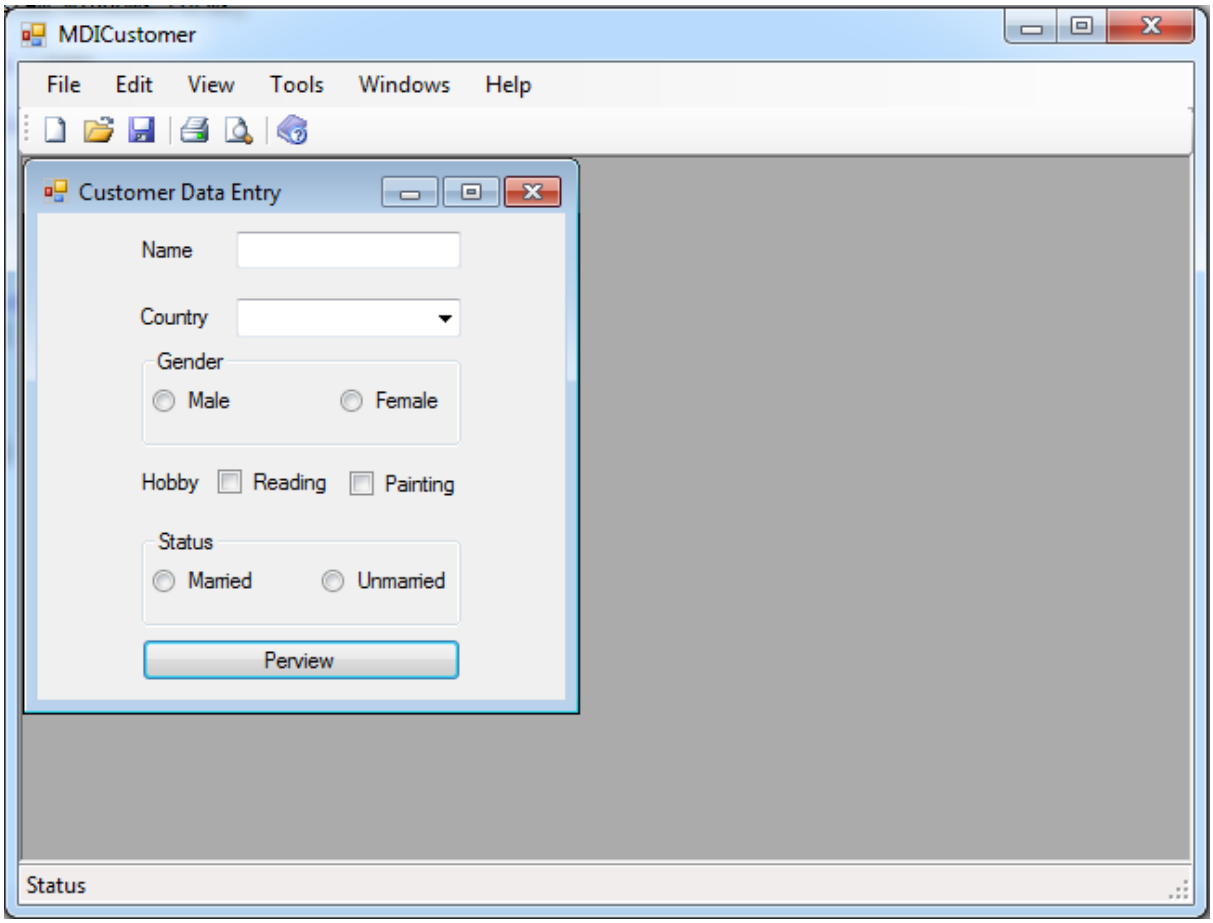
In MDI Parent form:

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

```
    frmCustomerDataEntry obj = new frmCustomerDataEntry();
    obj.MdiParent = this;
    obj.Show();
}
```

In program.cs:

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MDICustomer());
}
```



REUSING CODE BY CREATING CLASSES AND OBJECTS AND FORM VALIDATION

1. An important feature of programming is to avoid the reusability of code.
2. To write the validations of different types, create a new class.

3. **View → Solution Explorer → Select your project → Right Click → Add → New Item →** Add a new **Class** and name it as **CustomerValidation.cs**
4. In **CustomerValidation.cs** you can define various functions with different names and parameters. An example is given below -

In project Validations:

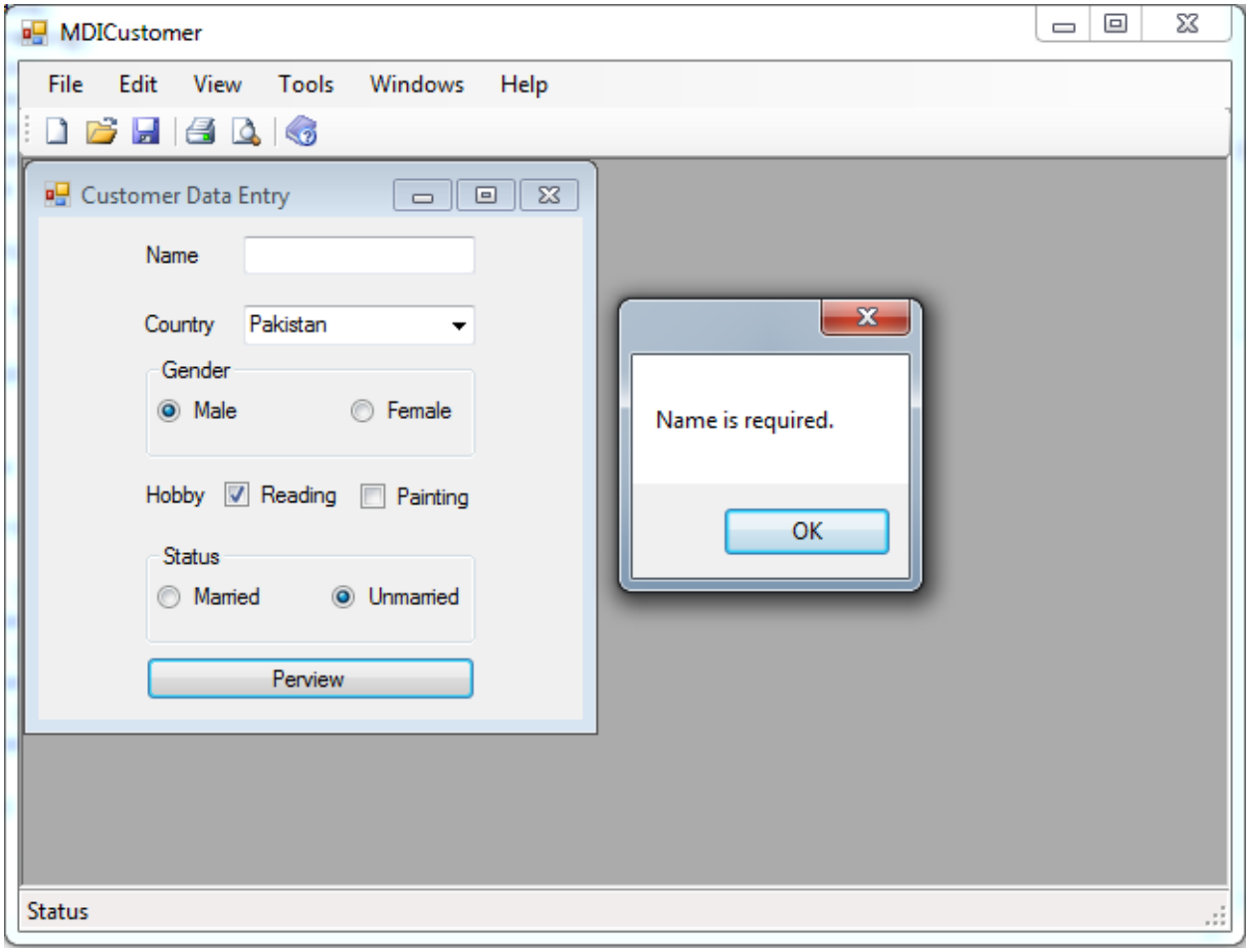
```
namespace Validations
{
    public class CustomerValidation
    {
        public void CheckCustomerName(string CustomerName)
        {
            if (CustomerName.Length > 10)
                throw new Exception("Name should be within 10 characters.");
            else if (CustomerName == "")
                throw new Exception("Name is required.");
        }

         }
     }
 }
```

In frmCustomerDataEntry:

1. In the Preview button Click event we have write some code to specify that if the user does not enter any **Customer Name** then an error message should be displayed.
2. We have to call the **CheckCustomerName (string)** function in the **CustomerValidation** class.
3. This code is specified in the **try-catch block** below -

```
using Validations;

namespace Lab04
{
    public partial class frmCustomerDataEntry : Form
    {
        public frmCustomerDataEntry()
        {
            InitializeComponent();
        }

        private void btnPreview_Click(object sender, EventArgs e)
        {
            string Gender, Hobby, Status = "";

            if (radioMale.Checked) Gender = "Male";
            else Gender = "Female";
            if (chkReading.Checked) Hobby = "Reading";
            else Hobby = "Painting";
            if (radioMarried.Checked) Status = "Married";
            else Status = "Unmarried";
```

```
            try
            {
                CustomerValidation objVal = new CustomerValidation();
                objVal.CheckCustomerName(txtName.Text);

                frmCustomerPreview objPreview = new frmCustomerPreview();
                objPreview.SetValues(txtName.Text, cmbCountry.Text,
                                     Gender, Hobby, Status);

                objPreview.Show();
            }
            catch (Exception ex)
            {

                MessageBox.Show(ex.Message.ToString());
            }
        }
    }
}
```



CONNECTING TO SQL SERVER, GETTING DATA AND GETTING ACQUAINTED WITH ADO.NET COMPONENTS, LIKE CONNECTION, COMMAND AND DATA READER

- As **C#** is a Microsoft Product, it is better to use a Microsoft Data management software which is **SQL Server.**
- **ADO.NET (ActiveX Data Object)** helps to connect the **User Interface** to the **Database** and also sends and retrieves data from SQL Server
- **To Create a Database connection you have to follow the steps below –**
    1. **View → Server Explorer → Select Data Connections → Right Click → Create New SQL Server Database → Server Name: ".\sqlexpress " → Give a Database Name: "CustomerDB" → Ok**
    2. **View → ToolBox → Data → Data GridView → Drag it to the frmCustomerDataEntry** form and name it as **dtgCustomer.**
    3. **View → Server Explorer → Here your connection is shown under Data Connection.** Select that new connection → **Right Click → Properties → Connection String (This string is needed in the code below)**

    4. To create a new Table in the Database you have to select your new connection and under it there is an option **Tables. Right Click on the Tables → Create New Table**

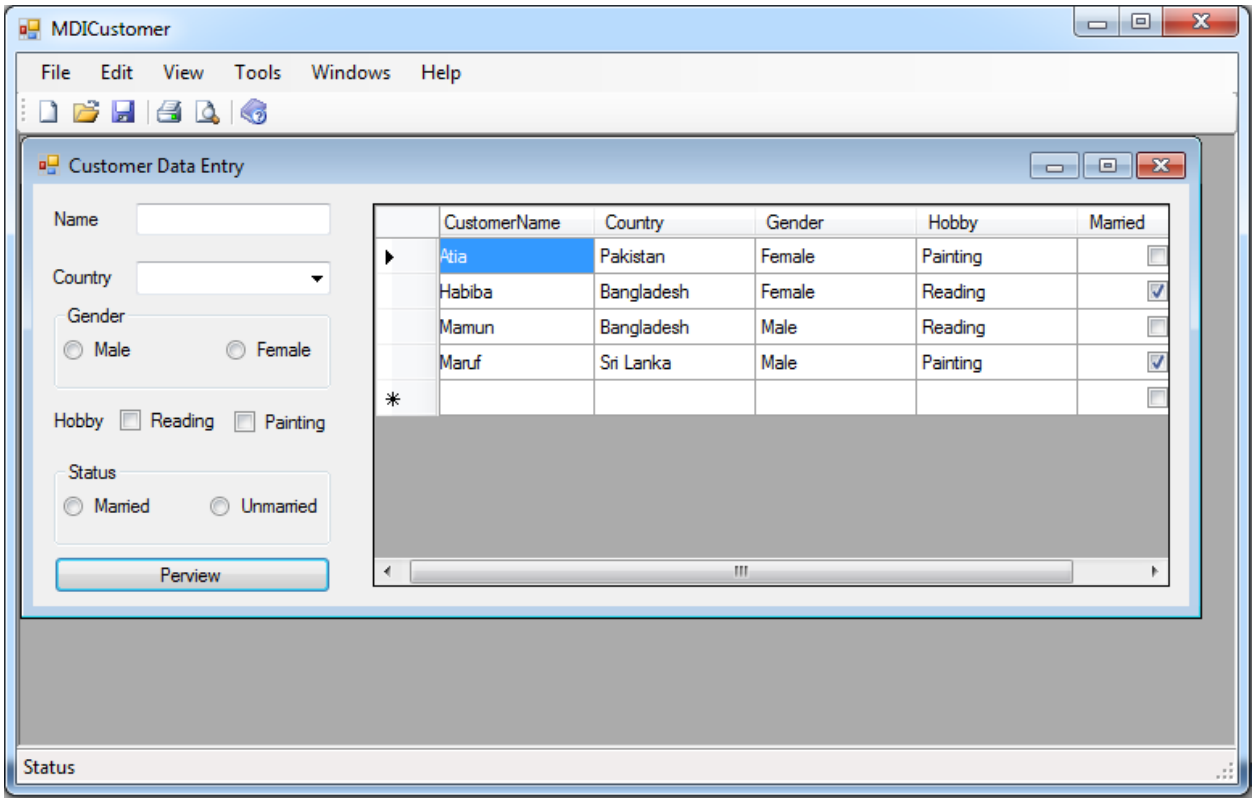
    5. Save this and give a **name** (here **CustomerDB** is given) to the **Table**

    6. Now that your Table has been created, you can add different columns: Select your **Table → Right Click → Show Table Definition → Now add column names with respective datatypes**

    7. You can add data in your Table: Select your **Table → Right Click → Show Table Data**

```
private void frmCustomerDataEntry_Load(object sender, EventArgs e)
{
    loadCustomer();
}

private void loadCustomer()
{
    // Open a Connection
    string strConnection = "Data Source=.\\sqlexpress;Initial Catalog=CustomerDB;"
                           + "Integrated Security=True";
    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "Select * From CustomerTable";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);
```

```
    // Bind Data with UI
    DataSet objDataSet = new DataSet();
    SqlDataAdapter objAdapter = new SqlDataAdapter(objCommand);
    objAdapter.Fill(objDataSet);
    dtgCustomer.DataSource = objDataSet.Tables[0];

    // Close the Connection
    objConnection.Close();
}
```

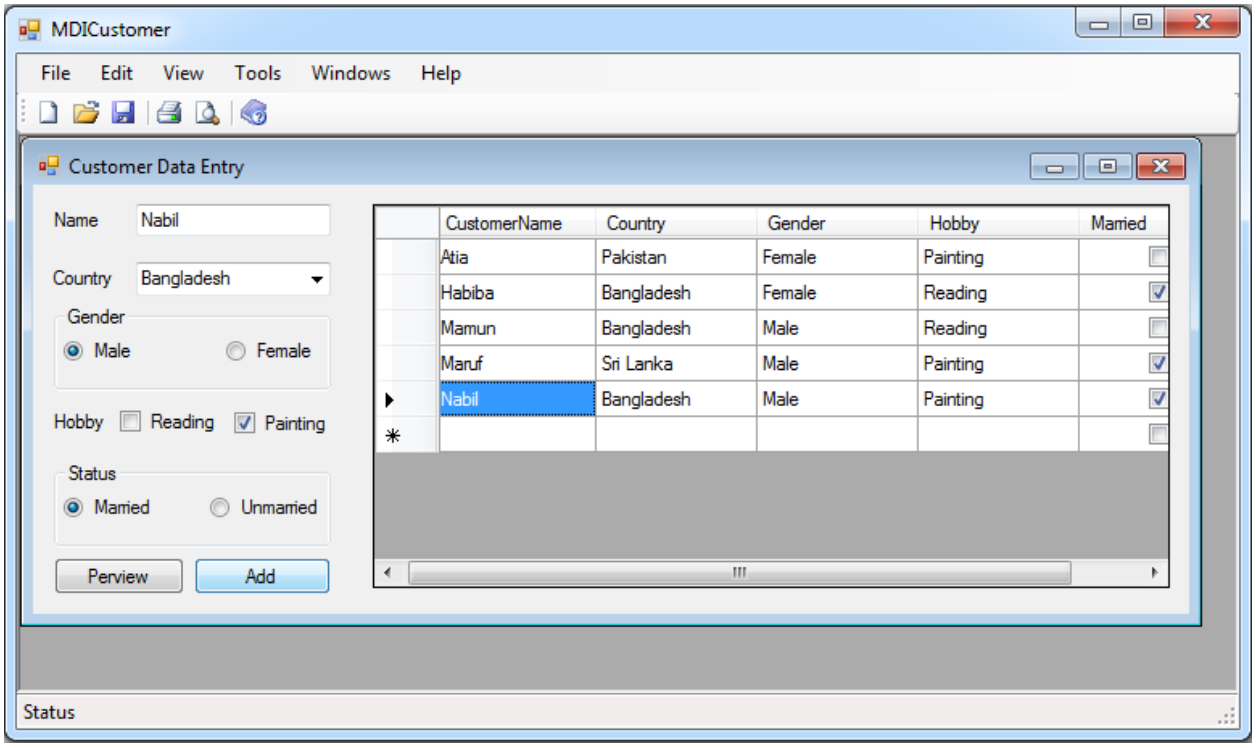


LAB 05: INSERTING INTO DATABASE AND DYNAMIC CONNECTION STRING

INSERTING DATA FROM THE UI TO SQL SERVER

```
private void btnAdd_Click(object sender, EventArgs e)
{
    string Gender, Hobby, Status = "";

    if (radioMale.Checked) Gender = "Male";
    else Gender = "Female";
    if (chkReading.Checked) Hobby = "Reading";
    else Hobby = "Painting";
    if (radioMarried.Checked) Status = "1";
    else Status = "0";
```

```
    // Open a Connection
    string strConnection = "Data Source=.\\sqlexpress;Initial Catalog=CustomerDB;"
                           + "Integrated Security=True";
    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "insert into CustomerTable values('"+txtName.Text+"', '"
                                                            +cmbCountry.Text+"','"
                                                            +Gender+"', '"
                                                            +Hobby+"', "
                                                            +Status+" )";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);
    objCommand.ExecuteNonQuery();

    // Close the Connection
    objConnection.Close();

    loadCustomer();
}
```



MAKING THE CONNECTION STRING CONFIGURABLE BY STORING IT IN A XML CONFIGURATION FILES RATHER THAN HARD-CODING IT IN THE C# CODE

It is nothing but to save some of your configuration such as database connection string in a separate file so that you don't need to write

the long boring connection string every time you want to create a database connection in your program. Another advantage of using this is – if you want to change your connection/database/database name, you only have to make change in the .config file.

How to configure a web.config file?
You need to refer the 'web.config' file to your asp file. 'app.config' is generally used in windows applications. Follow the steps:
1. Delete any .config file you added previously to your web project.
2. Right click on your web project -> add Item -> web configuration file.
3. Copy your data connection string. (For the sake of the example, let us consider that (just for example) your connection string is- "Data Source=.\sqlexpress;Initial Catalog=CustomerDB;Integrated Security=True;Pooling=False")
4. Open your web.config file.
5. Delete everything inside it, then paste following lines there-

```
<?xml version="1.0"?>
<configuration>
<connectionStrings>
<add name="DBConn" connectionString="Data Source=.\sqlexpress;Initial
Catalog=CustomerDB;Integrated Security=True;Pooling=False"/>
</connectionStrings>
<system.web>
<compilation debug="true" targetFramework="4.0" />
</system.web>

</configuration>
```

6. Make sure you have changed the connectionString to your database connection string.
7. Now, suppose that, you are creating a database connection in a file named program.cs, which is under project named project1.
8. right click on project1 -> add reference-> .NET -> System.configuration
9. Now, add following line to your program.cs file-

```
using System.Configuration;
```

10. Finally you are ready to take the advantage of your web.config file. Previously, when you wanted to create a connection, you had to write that long connection string in your program.cs file, but now you can write it dynamically like bellow-

```
string connectionString = ConfigurationManager.ConnectionStrings["DBConn"].ToString();
```

** Remember, DBConn is the variable you used to store the connection string in the web.config file. you can choose any name instead if you wish.

LAB 06: UPDATING AND DELETING FROM DATABASE

SELECTING DATA WHEN A USER CLICKS ON THE DATA GRID HYPER LINK

```
private void dtgCellClick(object sender, DataGridViewCellEventArgs e)
{
    clearForm();
    string CustomerName = dtgCustomer.Rows[e.RowIndex].Cells[0].Value.ToString();
    displayCustomer(CustomerName);
}
private void clearForm()
{
    txtName.Text = "";
    cmbCountry.Text = "";
    radioMale.Checked = false;
    radioFemale.Checked = false;
    chkPainting.Checked = false;
    chkReading.Checked = false;
    radioMarried.Checked = false;
    radioUnmarried.Checked = false;
}
```

UPDATING DATA BY USING THE UPDATE SQL COMMAND USING THE ADO.NET COMMAND OBJECT

```
private void displayCustomer(string strCustomer)
{
    // Open a Connection
    string strConnection = ConfigurationManager.ConnectionStrings["DBConn"].ToString();
    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "Select * From CustomerTable where CustomerName =
'"+strCustomer+"'";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);

    // Bind Data with UI
    DataSet objDataSet = new DataSet();
    SqlDataAdapter objAdapter = new SqlDataAdapter(objCommand);
```

```
    objAdapter.Fill(objDataSet);

    objConnection.Close();

    txtName.Text = objDataSet.Tables[0].Rows[0][0].ToString();
    cmbCountry.Text = objDataSet.Tables[0].Rows[0][1].ToString();
    string Gender = objDataSet.Tables[0].Rows[0][2].ToString();
    if (Gender.Equals("Male")) radioMale.Checked = true;
    else radioFemale.Checked = true;
    string Hobby = objDataSet.Tables[0].Rows[0][3].ToString();
    if (Hobby.Equals("Reading")) chkReading.Checked = true;
    else chkPainting.Checked = true;
    string Married = objDataSet.Tables[0].Rows[0][4].ToString();
    if (Married.Equals("True")) radioMarried.Checked = true;
    else radioUnmarried.Checked = true;
}
```

IMPLEMENTING DELETE FUNCTIONALITY BY WRITING THE DELETE SQL COMMAND IN THE ADO.NET COMMAND OBJECT

```
private void btnDelete_Click(object sender, EventArgs e)
{
    // Open a Connection
    string strConnection = ConfigurationManager.ConnectionStrings["DBConn"].ToString();

    SqlConnection objConnection = new SqlConnection(strConnection);
    objConnection.Open();

    // Fire a Command
    string strCommand = "Delete from CustomerTable where CustomerName =
'"+txtName.Text+"'";
    SqlCommand objCommand = new SqlCommand(strCommand, objConnection);
    objCommand.ExecuteNonQuery();

    // Close the Connection
    objConnection.Close();
    clearForm();
    loadCustomer();
}
```

LAB 07: TWO-LAYERED ARCHITECTURE

BREAKING THE PROJECT INTO A 2-LAYERED ARCHITECTURE, THE FIRST LAYER WILL HAVE UI + BUSINESS LOGIC AND THE SECOND LAYER WILL HAVE THE SQL PART I.E. DATA ACCESS.

In this lab we will learn how to organize our codes into two separate layers/projects. Our goal will be to separate the User Interface of

our project from the database related codes; so that we can achieve following benefits:

1) A cleaner, easier to read, more organized and more professional code.
2) Creation of a reusable data access library, which can later be called from different User interfaces of different projects.

The project we will use to show this example is CustomerDataEntry project; which we were working on in the labs.

Step 01: Create a new project under the same **Solution** of your current project (in this example, CustomerDataEntry solution). Don't select a windows/web form project, rather select class library. We are going to name it as DataAccess.
Step 02: A default class is created inside our DataAccess project. For convenience, we are going to name it as clsSqlServer.
Step 03: Now, we will gradually shift all our database related code from the user interface form to the newly created clsSqlServer class. We will create a function named getCustomerData inside the clsSqlServer.cs which will get all the customers' information from database and will return the result to the caller class, in this case to our user interface class. The function will look like this-

```
public DataSet getCustomerData()
{
string strConnection = "your database connection string";
SqlConnection objConnection = new SqlConnection(strConnection);
objConnection.Open();
SqlCommand objCommand = new SqlCommand("select * from CustomerTable", objConnection);
SqlDataAdapter objAdapter = new SqlDataAdapter(objCommand);
DataSet ds = new DataSet();
objAdapter.Fill(ds);
return ds;
}
```

Step 04: Inside the load function of our user interface form, now all we have to do is call the getCustomerData() function of clsSqlServer class of DataAccess project. But you can't simply call a function of a class that does not belong to the same project. You have to give a reference to the DataAccess project from your Starting project (from where you are calling the getCustomerData function).

To do that, right click on the reference folder of the caller project (in this example, CustomerDataEntry project)-> Add Reference -> Projects -> DataAccess -> Ok.

Step 05: Inside the user interface form, add the namespace of DataAccess, like this->

```
using DataAccess;
```

Step 06: Now you are all set to call the getCustomerData() function from your user interface form. In the load function, delete all previous database related code and add following codes-

```
clsSqlServer obj = new clsCustomer();// declaring an object of clsSqlServer
class
DataSet ds = clsSqlServer.getCustomerData();//getting all the data from
database via DataAccess layer
dtgCustomer.DataSource = ds.Tables[0];//setting the datasource of your data
grid
```

Step 07: In this way, you can gradually shift all the database related code (insert/update/delete/search) from your user interface to the DataAccess layer.

LAB 08: CREATING ASP.NET WEB APPLICATION

CREATING AN ASP.NET WEB APPLICATION UI THAT WILL INTERACT WITH THE CUSTOMER DATA ENTRY DATABASE

CONNECTING THE ASP.NET WEB APPLICATION UI WITH THE DATA ACCESS LAYER

IMPLEMENTING INSERT, UPDATE AND DELETE FUNCTIONALITY FOR THE ASP.NET WEB APPLICATION UI.

LAB 09: THREE-LAYERED ARCHITECTURE

CONVERTING THE 2-LAYERED ARCHITECTURE TO 3-LAYERED ARCHITECTURE (3 LAYERS: UI, BUSINESS LOGIC AND DATA ACCESS)

LAB 10: SECURITY

IMPLEMENTING SECURITY IN WINDOWS APPLICATIONS

IMPLEMENTING WEB SECURITY USING FORMS AUTHENTICATION

LAB 11: VALIDATION AND CRYSTAL REPORT

IMPLEMENTING EMAIL VALIDATION EASILY USING A REGULAR EXPRESSION (REGEX)

CREATING CRYSTAL REPORTS

The steps to create a crystal report are almost same in windows and web application. I will use a "/" sign to differentiate between windows and web application.

If you are using Visual Studio 2005, please go here.
Others, you should first install crystal report software if not already. The link I am providing here may ask you to register first, just fill up the form; it will take a minute or two to register. Click here to download and install SAP Crystal Report.**(Very Important: don't download the .msi file, download the .exe)**

0. Close and restart your Visual Studio.
1. Create a new windows form project/web project. (File->New->project->windows form project/web project)
2. Go to Form1.cs/default.aspx in design view, drag and drop a CrystalReportViewer from toolbox; name it as crViewer.
(If you don't see CrystalReportViewer in the toolbox, right click on your project name in the solution explorer; select properties; select .NET Framework 4.0 or .NET Framework 3.5)
3. View->Solution Explorer; right click on your project name; add new item; add a Dataset; name it as dtsCustomer; the gray colored Dataset page will be opened.
4. View->Server Explorer, select the table you wish to make a report from (in my case it was the famous CustomerTable); drag and drop it to your gray colored Dataset page.
5. right click on project name; add new item; add crystal report; name it as crCustomer.

6. Crystal reports gallery should be opened; select “using the report wizard”; click OK; Project Data -> ADO.NET Datasets -> CustomerTable(or the table you have dragged to the dataset a minute ago); click the “>” sign; click next; expand your table by clicking the + sign; click >> sign to get all your table columns in the right side; next; next; next; finish.
7. Go to Form1.cs/default.aspx.cs (code view) then add following namespace to your code-

```
using System.Data.SqlClient;
```

8. Copy and paste following code inside the Form1_Load()/Page_Load function-

```
//step1: fill your dataset with data from database
SqlConnection con = new SqlConnection(“Data Source=.\\sqlexpress;Initial Catalog=CustomerDB;Integrated Security=True”);// you should replace this code with your data connection string
con.Open();
SqlCommand com = new SqlCommand(“select * from CustomerTable”,con);
SqlDataAdapter adapter = new SqlDataAdapter(com);
dtsCustomer dts = new dtsCustomer(); // you have created dtsCustomer just a while ago in your project
adapter.Fill(dts.CustomerTable);// now your dtsCustomer dataset is filled with Customer information
//step2: now set the dtsCustomer dataset as your crystal report’s datasource
crCustomer crs = new crCustomer(); // crCustomer is your Crystal Report’s class
crs.SetDataSource(dts);
//step3: now bind this crystal report with your CrystalReportViewer crViewer
crViewer.ReportSource = crs;
```

9. That should do. Press Ctrl+F5 to run your Crystal Report project on the windows form/browser.

10. If you get a FileException error, then go to your app.config file. Add useLegacyV2RuntimeActivationPolicy=”true” in the <startup > tag. That means, your startup tag should look like this -

```
<startup useLegacyV2RuntimeActivationPolicy=”true”><supportedRuntime version=”v4.0” sku=”.NETFramework,Version=v4.0”/></startup>
```