## MVP(Model-View-Presenter) Pattern in C#

MVP is one of the design pattern which separates the visual display from the even handling. There are three main components in MVP pattern, those are Model, View, and Presenter. In this article we will discuss about the Model-View-Presenter pattern in C# with simple example.

Model handles the data level operations. That means Model is the business level or domain level object which connects the database and will perform read & write operations.

View is responsible for displaying controls on UI. To make View testable create the View interface and let’s view implementation class implements this interface. User interacts with the View for any operation.

Presenter responsible for handling any events generated in the View by implementation the View interface and calls the Model. Presenter sits between View & Model, it passes the user inputs from View to Model and passes the output from Model to View.

Open Microsoft Visual Studio 2013 => Create C# Windows application. Add three folders named as Model, View, and Presenter.

Let’s add some interface for Model and implement this interface in Model class.

namespace MVPExp.Model

{

public interface IEmployeeModel

{

string GetEmployeeName(string sEmployeeId);

}

}

namespace MVPExp.Model

{

public class EmployeeModel : IEmployeeModel

{

public EmployeeModel() { }

public string GetEmployeeName(string sEmployeeId)

{

return "Raj";

}

}

}

Now create IEmployeeView interface for View as shown below.

namespace MVPExp.View

{

public interface IEmployeeView

{

string EmployeeId { get; set; }

string EmployeeName { get; set; }

}

}

Create Presenter class EmployeePresenter which implements the View interface IEmployeeView as shown below. This presenter class calls Model class EmployeeModel by passing the inputs from View class.

using MVPExp.Model;

using MVPExp.View;

namespace MVPExp.Presenter

{

public class EmployeePresenter

{

IEmployeeView empView;

private Form1 form1;

public EmployeePresenter(IEmployeeView view)

{

empView = view;

}

public string GetEmployeeName()

{

EmployeeModel model = new EmployeeModel();

empView.EmployeeName = model.GetEmployeeName(empView.EmployeeId).ToString();

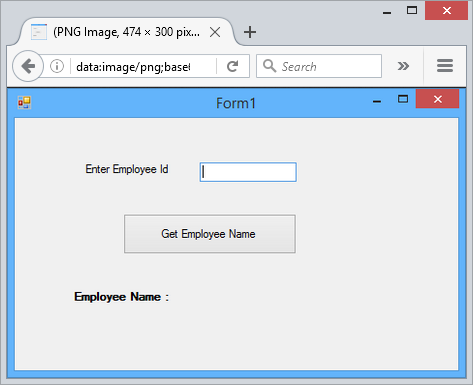
return empView.EmployeeName.ToString();

}

}

}

Finally create View, which means UI. Add controls to the Windows form as shown below.



In code-behind implement the view interface IEmployeeView and call the presenter class as shown below.

using MVPExp.Presenter;

using System;

using System.Windows.Forms;

namespace MVPExp.View

{

public partial class Form1 : Form, IEmployeeView

{

public Form1()

{

InitializeComponent();

}

private void btn1\_Click(object sender, EventArgs e)

{

EmployeePresenter presenter = new EmployeePresenter(this);

presenter.GetEmployeeName();

}

public string EmployeeId

{

get { return txtEmpId.Text; }

set { txtEmpId.Text = value; }

}

public string EmployeeName

{

get { return lblEmpName.Text; }

set { lblEmpName.Text = value; }

}

}

}

As shown above we implemented the IEmployeeView interface properties EmployeeId & EmployeeName by assigning the txtEmpId & lblEmpName controls Text value.

Run the application, enter the Id for EmployeeId text box and click on button. It calls the presenter by passing Employee Id value. Presenter class calls the Model class with Employee Id value, in return Model class passes the Employee Name to Presenter. Presenter passes the Employee Name value to View. Here Presenter class completely independent of View by implementing IEmployeeView interface.