Model-View-Presenter

(MVP)

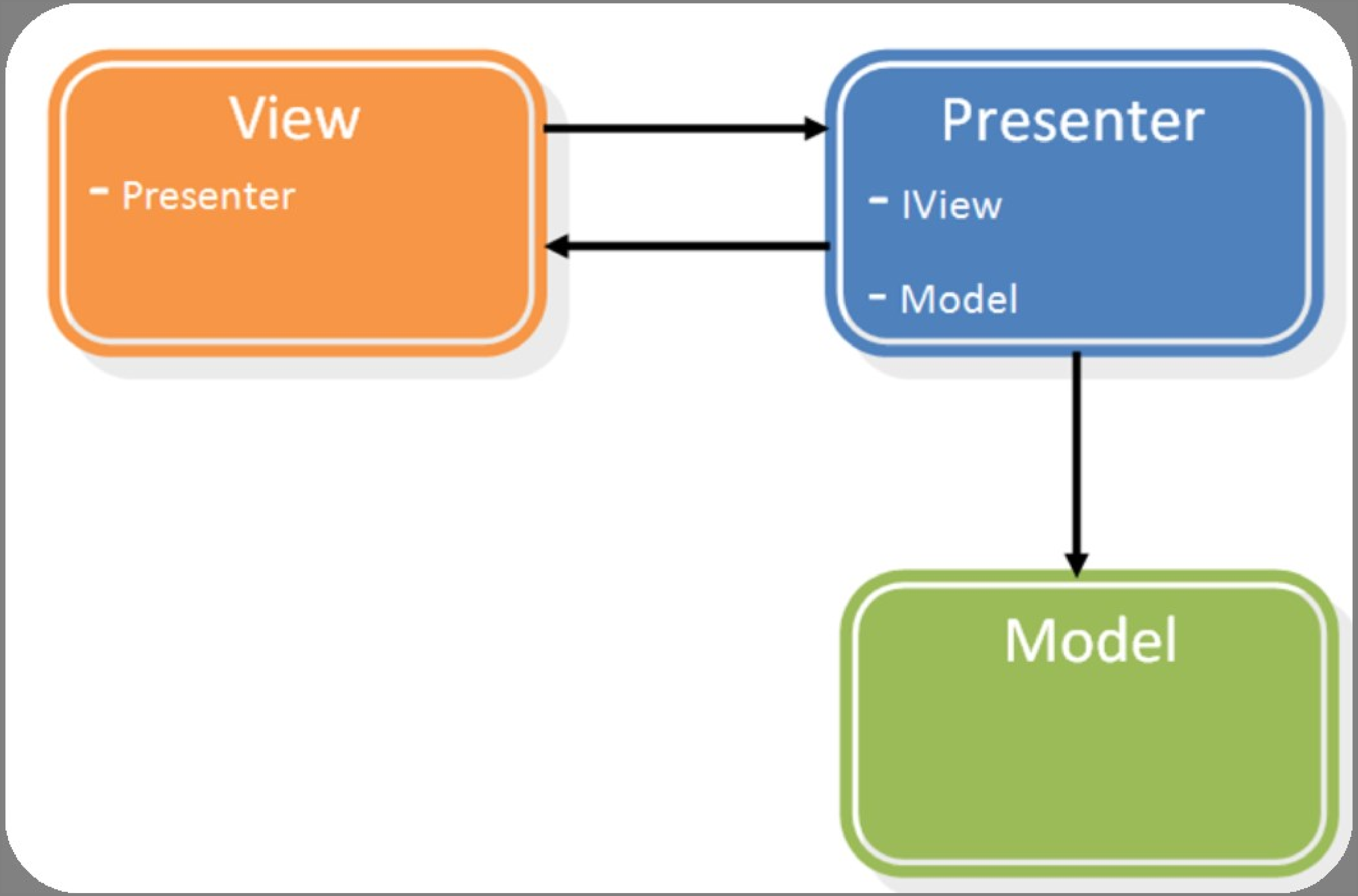
What Is MVP

- MODEL VIEW PRESENTER (MVP) IS A SOFTWARE DESIGN PATTERN WHICH ESSENTIALLY ISOLATES THE USER INTERFACE FROM THE BUSINESS LOGIC. MVP IS DERIVED FROM THE MODEL VIEW CONTROLLER (MVC) PATTERN, AND ORIGINALLY CONCEIVED BY THE RENOWNED AGILE SOFTWARE ARCHITECT, MARTIN FOWLER.

- THE PRINCIPAL BEHIND THE MVP PATTERN IS THAT AN IMPLEMENTING APPLICATION SHOULD BE SPLIT INTO THREE CORE COMPONENTS; MODEL, VIEW AND PRESENTER:

Principal

- THE **MODEL** COMPONENT ENCAPSULATES ALL BUSINESS LOGIC AND DATA IN THE APPLICATION. THIS MAY BE A DATABASE TRANSACTION OR A CALL TO A WEB SERVICE, ETC.
- THE **VIEW** COMPONENT REPRESENTS THE APPLICATION'S PRESENTATION LAYER (USER INTERFACE); THIS MAY BE A STANDARD WIN FORMS CLIENT, AN ASP.NET WEB PART OR MOBILE CLIENT. IN THE MVP PATTERN, THE VIEW SHOULD BE SIMPLISTIC AND RESPONSIBLE FOR RENDERING AND ACCEPTING USER INPUT ONLY.
- THE **PRESENTER** COMPONENT IS RESPONSIBLE FOR ORCHESTRATING ALL THE APPLICATION'S USE CASES. FOR EXAMPLE A SAMPLE OPERATION WOULD INVOLVE; TAKING USER INPUT FROM THE VIEW, INVOKING OPERATIONS ON THE MODEL AND IF NEEDED, SETTING DATA IN THE VIEW TO INDICATE THE OPERATION'S RESULT.


View
- Presenter
Presenter
- IView
- Model
Model

The advantages of implementing the MVP pattern in a project with a presentation tier are:

- **ISOLATION OF USER INTERFACE FROM BUSINESS TIER**
- **EASILY INTERCHANGEABLE VIEWS (USER INTERFACES)**
- **ABILITY TO TEST ALL CODE IN THE SOLUTION (EXCLUDING VISUAL PRESENTATION AND INTERACTION)**

Example Search Application

- USER ENTERS A SEARCH CRITERIA (E.G. “TEST”)
- THE USER INITIATES A SEARCH IN THE UI (E.G. BUTTON CLICK)
- THE VIEW COMPONENT HANDLES THE SEARCH EVENT, AND INVOKES THE ‘SEARCH()’ FUNCTION ON THE PRESENTER
- THE PRESENTER EXTRACTS THE SEARCH CRITERIA STRING FROM THE VIEW’S ‘SEARCHCRITERIA’ PROPERTY
- THE PRESENTER INVOKES THE BUSINESS OPERATION ‘BUSINESS SEARCH’ ON THE MODEL, PASSING THE SEARCH CRITERIA AS AN ARGUMENT
- THE MODEL RETURNS ALL MATCHING RESULTS FOR THE GIVEN SEARCH CRITERIA
- THE PRESENTER SETS THESE RESULTS IN THE VIEW COMPONENT’S ‘RESULTS’ PROPERTY

Model

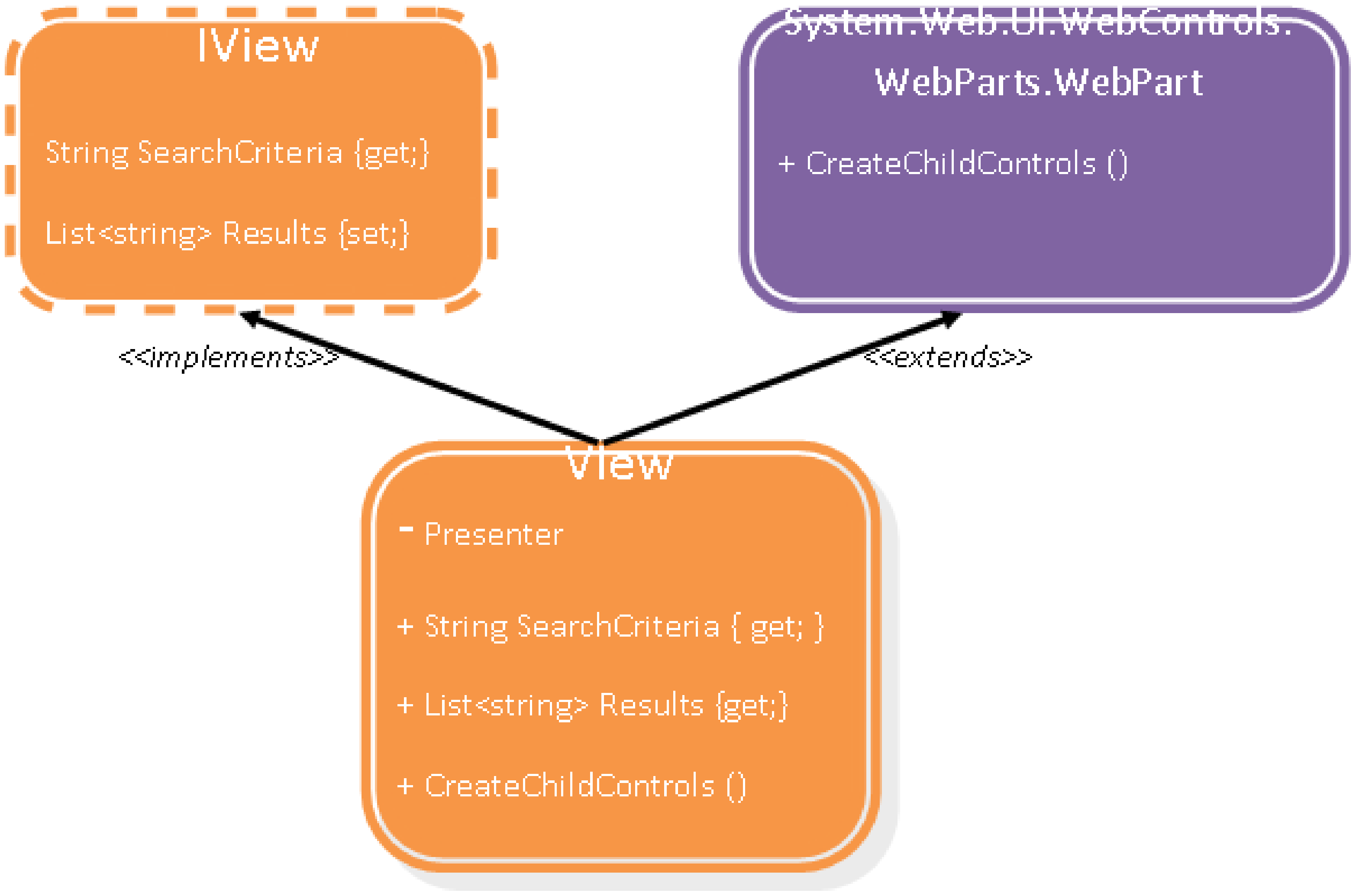
- THE MVP PATTERN IS CHIEFLY CONCERNED WITH ISOLATING THE USER INTERFACE FROM THE BUSINESS LOGIC AND AS SUCH THE MODEL COMPONENT'S DESIGN IS NOT SPECIFIED IN ANY DETAIL.
- SINCE THE PRESENTATION COMPONENT IS RESPONSIBLE FOR ORCHESTRATING BUSINESS USE CASES, THE PRESENTATION COMPONENT CONTAINS A REFERENCE TO BOTH THE VIEW AND MODEL COMPONENTS.
- HOWEVER IT IS GENERALLY ACCEPTED AS GOOD PRACTICE TO DECOUPLE THE MODEL FROM THE PRESENTATION COMPONENT. TO DO THIS, IT IS RECOMMENDED TO CREATE AN INTERFACE FOR THE MODEL WHICH DEFINES ALL THE BUSINESS LOGIC OPERATIONS CONTAINED IN THE MODEL. THEN USE THE 'FACTORY METHOD' PATTERN (A CREATIONAL PATTERN) TO RETURN A CONCRETE IMPLEMENTATION OF THE MODEL FOR USE IN THE PRESENTATION COMPONENT. THIS ALLOWS THE MODEL TO BE INTERCHANGED WITHOUT ANY MODIFICATION TO THE PRESENTATION COMPONENT.

View

- THE VIEW COMPONENT REPRESENTS THE PRESENTATION LAYER AND CAN BE IMPLEMENTED IN A VARIETY OF UI TECHNOLOGIES (WINDOWS FORMS, WEB-PAGE, WEB-PART, SILVERLIGHT ETC) ON A RANGE OF PLATFORMS (CLIENT, WEB, MOBILE).
- THE VIEW'S RESPONSIBILITIES ARE LIMITED TO RENDERING ITSELF, ACCEPTING USER INPUT AND HANDLING USER EVENTS ON CONTROLS (E.G. BUTTON CLICKS).
- THE VIEW DOES NOT PERFORM ANY BUSINESS LOGIC, OR DIRECTLY INTERACT WITH THE MODEL. INSTEAD IT INVOKES METHODS ON THE PRESENTER. THE PRESENTER IN-TURN CALLS BUSINESS OPERATIONS ON THE MODEL AND THEN [IN SOME CASES] SETS PROPERTIES ON THE VIEW TO INDICATE THE OPERATION'S RESULT.
- BECAUSE THE VIEW IS DESIGNED TO BE FULLY INTERCHANGEABLE, THE VIEW COMPONENT IMPLEMENTS AN INTERFACE WHICH DEFINES THE METHODS AND PROPERTIES ALL CONCRETE VIEWS MUST IMPLEMENT.

- THE FOLLOWING DIAGRAM ILLUSTRATES THE ARCHITECTURE OF THE VIEW COMPONENT. TO BETTER UNDERSTAND HOW THE VIEW AND PRESENTER COMPONENTS INTERACT, THESE COMPONENTS' WILL BE DISCUSSED IN THE CONTEXT OF THE EXAMPLE SEARCH WEB-PART APPLICATION.

- THE PRECEDING DIAGRAM, THE VIEW COMPONENT'S INTERFACE 'IVIEW' DEFINES A PROPERTY 'SEARCHCRITERIA' OF TYPE STRING WITH READ PERMISSIONS AND A PROPERTY 'RESULTS' OF TYPE STRING COLLECTION WITH WRITE PERMISSIONS.


IView
String SearchCriteria {get;}
List<string> Results {set;}
System.Web.UI.WebControls.
WebParts.WebPart
+ CreateChildControls ()
<<implements>>
<<extends>>
View
- Presenter
+ String SearchCriteria { get; }
+ List<string> Results {get;}
+ CreateChildControls ()

```
public interface Iview

{
    string SearchCriteria { get; }

    List<string> Results { set; }
}
```

THE PRESENTER COMPONENT IS NEVER GIVEN A DIRECT REFERENCE TO THE VIEW COMPONENT'S CONCRETE IMPLEMENTATION. INSTEAD IT REFERENCES THE VIEW COMPONENT THROUGH THE VIEW INTERFACE 'IVIEW'. THIS ALLOWS THE VIEW IMPLEMENTATION TO BE CHANGED (E.G. FROM A WEB PART CONTROL TO A WINDOWS FORM CLIENT) WITHOUT THE PRESENTER BEING MODIFIED. THEREFORE WHEN THE VIEW'S INTERFACE IS DEVELOPED, THOUGHT SHOULD BE GIVEN AS TO HOW THE PRESENTER WILL INTERACT WITH THE VIEW'S DATA. FOR EXAMPLE, IN THE EXAMPLE SEARCH APPLICATION, THE PRESENTER WILL NEED TO RETRIEVE ('GET') THE USER'S SEARCH CRITERIA STRING AND ONCE THE SEARCH HAS BEEN PERFORMED (BY THE MODEL) THE RESULTANT OUTPUT MUST BE DISPLAYED TO THE USER ('SET').

- IT IS ALSO IMPORTANT TO DEVELOP VIEW INTERFACES USING ONLY FRAMEWORK TYPES OR BUSINESS TYPES (AS DEFINED IN THE MODEL), NOT PRESENTATION TECHNOLOGY-SPECIFIC TYPES. E.G. IT WOULD BE POSSIBLE TO DEFINE THE 'RESULTS' PROPERTY TO BE OF TYPE SYSTEM.WEB.UI.WEBCONTROLS.BULLETEDLIST, HOWEVER THIS IS SPECIFIC TO AN ASP.NET VIEWS AND WOULD BE INEFFICIENT TO IMPLEMENT ON A WINDOWS FORM CLIENT VIEW IMPLEMENTATION.

- THE FOLLOWING CODE EXTRACT CONTAINS ALL THE CODE REQUIRED TO IMPLEMENT A VERY BASIC SEARCH ASP.NET WEB PART THAT ALSO IMPLEMENTS THE IVIEW INTERFACE.

```
public class ViewImpl : WebPart, Iview
{
    private Presenter presenter;
    Label lblSearch = new Label() { Text = "Enter bank to search for: " };
    Button btnSearch = new Button() { Text = "Search" };
    BulletedList resultsList = new BulletedList();
    TextBox tbSearchCriteria = new TextBox();

    public ViewImpl()
    {
        this.presenter = new Presenter(this);
    }

    protected override void CreateChildControls()
    {
        this.Controls.Clear();
        this.btnSearch.Click += new EventHandler(btnSearch_Click);
        this.Controls.Add(this.lblSearch);
        this.Controls.Add(this.tbSearchCriteria);
        this.Controls.Add(this.btnSearch);
        this.Controls.Add(this.resultsList);
        this.ChildControlsCreated = true;
    }
    void btnSearch_Click(object sender, EventArgs e)
    {
        this.presenter.Search();
    }

    #region IView Members
    public string SearchCriteria
    {
        get { return this.tbSearchCriteria.Text; }
    }

    public List<string> Results
    {
        set
        {
            this.resultsList.Items.Clear();
            List<string> results = new List<string>(value);
            foreach (string r in results)
            {
                this.resultsList.Items.Add(r);
            }
        }
    }
    #endregion
}
```

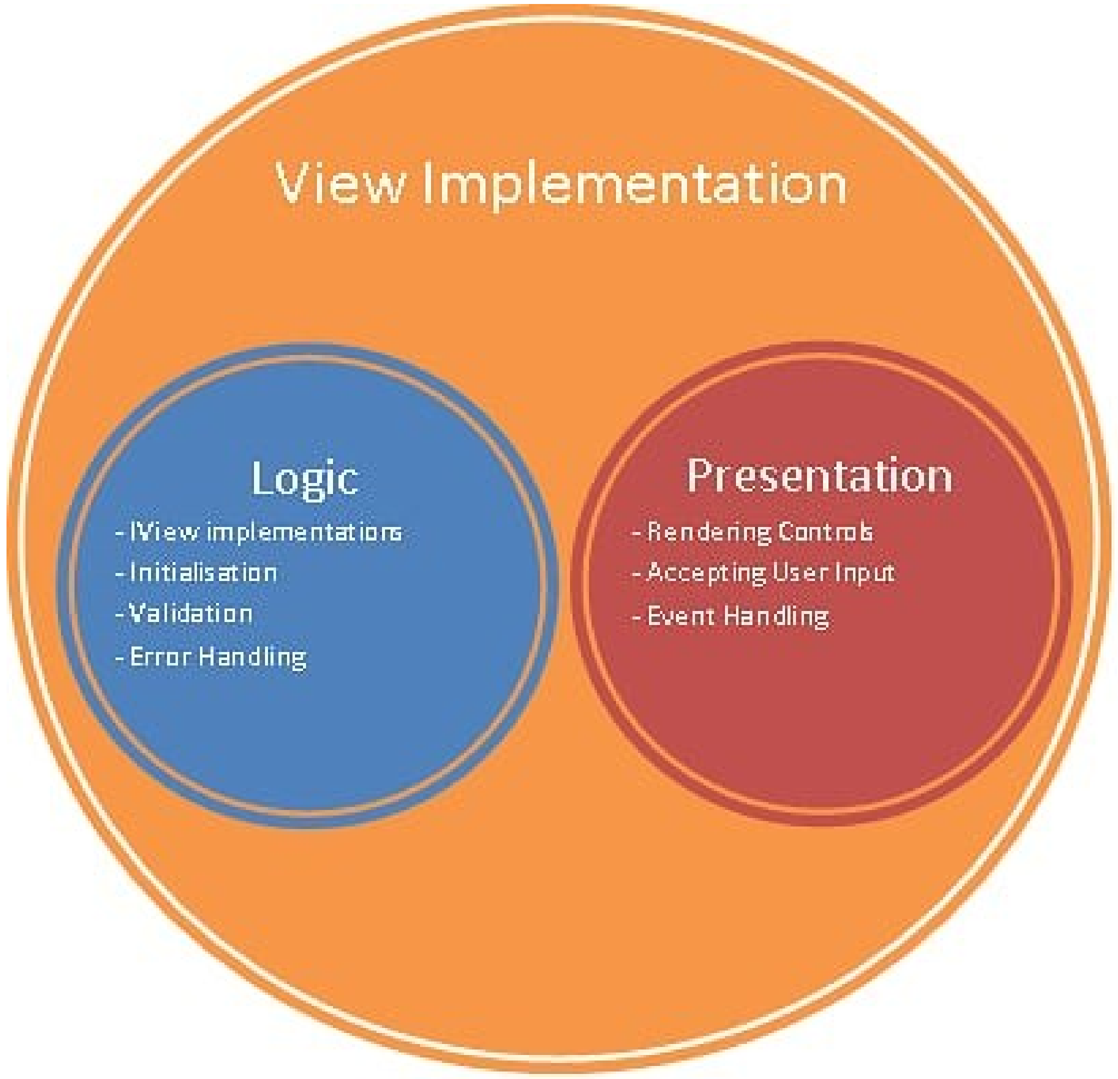
The following list dissects the structure and members of the example search web part view implementation:

- FIRSTLY, THE VARIABLES ARE DECLARED, INCLUDING A REFERENCE TO THE PRESENTER, ALSO A NUMBER OF UI CONTROLS.
- VIEWIMPL'S CONSTRUCTOR, INSTANTIATES THE PRESENTER, PASSING A REFERENCE TO ITSELF TO THE PRESENTER'S CONSTRUCTOR. THE PRESENTER'S CONSTRUCTOR SIGNATURE DEFINES A PARAMETER OF TYPE IVIEW, AND SINCE VIEWIMPL IMPLEMENTS IVIEW, THE SELF-REFERENCE CAN SAFELY BE PASSED. THIS ALLOWS THE PRESENTER TO INVOKE METHODS/PROPERTIES ON THE IVIEW INSTANCE AND VISA-VERSA ALLOWING THE VIEW TO INVOKE METHODS/PROPERTIES ON THE PRESENTER INSTANCE.

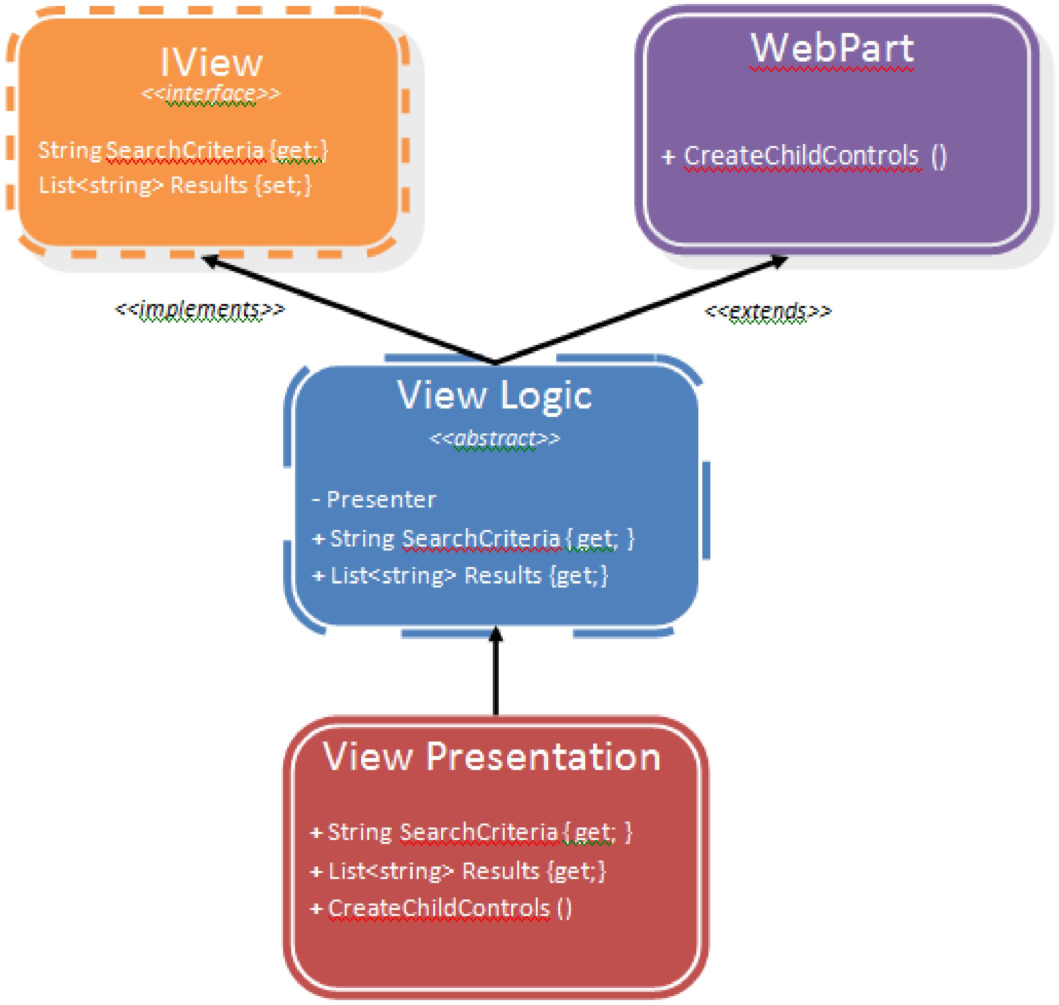
- VIEWIMPL OVERRIDES THE WEB PART METHOD ‘CREATECHILDCONTROLS()’ SO AS TO PERFORM CUSTOM RENDERING OF ITSELF AND ITS CHILDREN. THE ‘SEARCH’ BUTTON CONTROL ALSO REGISTERS A NEW EVENT HANDLER TO HANDLE BUTTON CLICKS.
- THE BUTTON CLICK EVENT HANDLER, DOES NOT PERFORM ANY LOGIC OR CALL ANY BUSINESS OPERATIONS, INSTEAD IT SIMPLY INVOKES A ‘SEARCH()’ METHOD ON THE PRESENTER (NOTE THAT THE METHOD DOES NOT TAKE ANY PARAMETERS).
- NEXT, THE IVIEW IMPLEMENTATIONS. THE ‘SEARCHCRITERIA’ PROPERTY RETURNS THE TEXT PROPERTY OF THE TEXTBOX INPUT CONTROL WHICH CONTAINS THE USER’S SEARCH CRITERIA. THE RESULTS PROPERTY POPULATES THE LIST CONTROL WITH THE DATA PASSED INTO IT.

Separation Of Concerns

- ALTHOUGH THE MVP PATTERN PRESCRIBES THE IMPLEMENTATION OF ONLY A VERY BASIC VIEW WITH LITTLE OR NO LOGIC, IT IS OFTEN DESIRABLE OR NECESSARY TO INCLUDE VALIDATION, ERROR HANDLING, INITIALIZATION, LOCALIZATION AND GENERAL HOUSE-KEEPING CODE. THESE OPERATIONS ARE SPECIFIC TO BE PRESENTATION-TECHNOLOGY BEING EMPLOYED AND AS SUCH ARE NOT SOMETHING THAT CAN BE HANDLED BY THE PRESENTER; HOWEVER THEY ARE ALSO NOT STRICTLY RELATED TO THE RENDERING OF THE VIEW.
- THEREFORE IN THE DEVELOPMENT OF THIS SIMPLE SEARCH EXAMPLE MVP APPLICATION, I DECIDED TO EXPERIMENT WITH A SEPARATION OF VIEW CONCERNS, WHEREBY THE VIEW WOULD BE SPLIT INTO A LOGIC TIER AND A PRESENTATIONAL TIER.


View Implementation
Logic
- IView implementations
- Initialisation
- Validation
- Error Handling
Presentation
- Rendering Controls
- Accepting User Input
- Event Handling

- IT IS IMPORTANT TO NOTE THAT IMPLEMENTING THE VIEW AS TWO SEPARATE CLASSES CAN LEAD TO MORE COMPLEX CODE AND REQUIRES GREATER DESIGN CONSIDERATION, AS IT MAY BE PRUDENT TO DECLARE CERTAIN IVIEW MEMBERS AS ABSTRACT IN THE LOGIC TIER AS THEY REQUIRE NO PRESENTATIONAL LOGIC AND SHOULD THEREFORE BE IMPLEMENTED IN THE PRESENTATIONAL TIER ONLY. THIS IS JUST ONE EXAMPLE OF A DESIGN CONSIDERATION THAT MUST BE TAKEN WHEN EMPLOYING THIS PATTERN.
- THEREFORE, THE FOLLOWING PATTERN IS ONLY REALLY SUITED TO COMPLEX VIEWS WHERE IMPLEMENTATION-SPECIFIC LOGIC IS ESSENTIAL.


IView
<<interface>>
String SearchCriteria {get;}
List<string> Results {set;}
WebPart
+ CreateChildControls ()
<<implements>>
<<extends>>
View Logic
<<abstract>>
- Presenter
+ String SearchCriteria { get; }
+ List<string> Results {get;}
View Presentation
+ String SearchCriteria { get; }
+ List<string> Results {get;}
+ CreateChildControls ()

Logic Tier

IN THE CONTEXT OF THE SEARCH WEB PART EXAMPLE, THE LOGIC TIER OF THE VIEW COMPONENT IS RESPONSIBLE FOR:

- IMPLEMENTATION OF THE IVIEW INTERFACE – THE BODY OF THESE PROPERTIES INCLUDES VALIDATION, ERROR HANDLING LOGIC ETC.
- INHERITING FROM WEBPART
- INITIALISATION – INSTANTIATING THE PRESENTER, SETTING UP RESOURCES SPECIFIC TO THE IMPLEMENTED TECHNOLOGY (E.G. RETRIEVING PARAMETERS FROM THE HTTP REQUEST OBJECT)
- VALIDATION – E.G. ENSURING THE RESULTS ASSIGNED TO THE RESULTS STRING COLLECTION ARE VALID
- ERROR HANDLING – E.G. HANDLING / THROWING EXCEPTIONS GENERATED DURING THE AFOREMENTIONED OPERATIONS

IT IS WORTH NOTING THAT ALTHOUGH THE LOGIC TIER INHERITS FROM SYSTEM.WEB.UI.WEBCONTROLS.WEBPARTS.WEBPART, IT IS MARKED AS AN ABSTRACT CLASS TO PREVENT INSTANTIATION. THIS IS BECAUSE IT IS ONLY RESPONSIBLE FOR IMPLEMENTATION-SPECIFIC LOGIC, NOT RENDERING OF ITSELF AND CHILD CONTROLS.

Presentational Tier

THE PRESENTATION TIER THEN INHERITS FROM THE LOGIC TIER, AND IS SOLELY RESPONSIBLE FOR ALL USER INTERFACE AND INTERACTION MATTERS. IN THE CONTEXT OF THIS SEARCH EXAMPLE, THESE RESPONSIBILITIES ARE:

- UI INITIALISATION – E.G. WEBPART TITLE, SIZE, POSITIONING ETC
- CREATION OF ALL CONSTITUENT CONTROLS (BUTTONS, TEXTBOXES, LABELS, LISTS ETC)
- RENDERING OF THESE CONTROLS – E.G. CREATECHILDCONTROLS()
- EVENT HANDLERS REGISTERED TO CONSTITUENT CONTROLS
- OVERRIDDEN IVIEW MEMBER IMPLEMENTATIONS – THE OVERRIDDEN IVIEW MEMBERS CALL THE LOGIC TIER'S IVIEW MEMBERS TO PERFORM VALIDATION ETC, AND THEN RENDER THE RESULTS TO THE RESPECTIVE UI CONTROLS.

Presenter

AS ILLUSTRATED IN THE VIEW IMPLEMENTATION C# CODE EXTRACT, THE VIEW (LIKE MANY APPLICATIONS) IS THE ENTRY-POINT FOR THE APPLICATION, AND IS RESPONSIBLE FOR CREATING THE PRESENTER WHICH IN-TURN CREATES THE MODEL. THE ONLY DIFFERENCE BETWEEN THIS EXAMPLE MVP SEARCH WEB PART AND A NON-MVP WEB PART IS THAT THE MVP VIEW CONTAINS NO BUSINESS LOGIC, APART FROM ONE CALL TO INVOKE THE SEARCH OPERATION ON THE PRESENTER INSTANCE. APART FROM THIS, IT IS SOLELY DEDICATED TO THE RENDERING OF ITS CONSTITUENT UI CONTROLS AND THE GETTING AND SETTING OF THE DATA THEREIN.

```
public class Presenter

{
    private readonly IView view;
    public Presenter(IView viewImpl)
    {
        this.view = viewImpl;
    }

    public void Search()
    {
        this.view.Results = Model.Search(this.view.SearchCriteria);
    }
}
```

THE PRESENTER IN THE EXAMPLE SEARCH APPLICATION (SEE ABOVE CODE EXTRACT) IS VERY SIMPLE AND CONTAINS A REFERENCE TO A VIEW (THROUGH THE IVIEW CONTRACT), A CONSTRUCTOR WHICH TAKES AN IVIEW REFERENCE AS ITS SINGLE PARAMETER AND A SINGLE METHOD ‘SEARCH()’.

```
public class Presenter

{
    private readonly IView view;
    public Presenter(IView viewImpl)
    {
        this.view = viewImpl;
    }

    public void Search()
    {
        this.view.Results = Model.Search(this.view.SearchCriteria);
    }
}
```

THE 'SEARCH()' METHOD IN THIS EXAMPLE, ILLUSTRATES HOW THE PRESENTER INTERACTS WITH THE VIEW. AS SHOWN IN THE CODE EXTRACT, THE METHOD DOES NOT TAKE ANY ARGUMENTS (SUCH AS THE USER'S 'SEARCH CRITERIA'). INSTEAD IT DIRECTLY OBTAINS THE SEARCH CRITERIA STRING FROM THE VIEW IMPLEMENTATION BECAUSE THE PRESENTER KNOWS THAT ANY VIEW IMPLEMENTATION MUST CONTAIN THE READ-ONLY PROPERTY 'SEARCHCRITERIA' OF TYPE STRING. SIMILARLY, INSTEAD OF RETURNING THE RESULTS OF THE SEARCH THROUGH A RETURN TYPE (THE 'SEARCH()' METHOD SIGNATURE DEFINES A RETURN TYPE OF VOID), IT DIRECTLY SETS THE RESULTS TO THE WRITE-ONLY PROPERTY 'RESULTS' OF TYPE STRING COLLECTION.

```
public class Presenter

{
    private readonly IView view;
    public Presenter(IView viewImpl)
    {
        this.view = viewImpl;
    }

    public void Search()
    {
        this.view.Results = Model.Search(this.view.SearchCriteria);
    }
}
```

NOTICE ALSO, THAT THE ‘SEARCH()’ METHOD DOES NOT PERFORM THE SEARCH OPERATION ITSELF, INSTEAD IT INVOKES THE MODEL TO PERFORM THIS TASK SINCE ITS PRIMARY RESPONSIBILITY IS TO EXECUTE BUSINESS OPERATIONS.

- THIS EXAMPLE PRESENTER CLEARLY ILLUSTRATES THE PRESENTER'S RELATIONSHIP WITH BOTH THE VIEW AND THE MODEL. IT DOES NOT KNOW WHAT VIEW IT IS RETRIEVING AND SETTING DATA TO, IT IS ONLY AWARE OF THE IVIEW INTERFACE WHICH ACTS AS A CONTRACT BETWEEN THE TWO COMPONENTS. SIMILARLY, THE PRESENTER DOES NOT NEED TO KNOW WHICH MODEL IT IS INVOKING ON BEHALF OF THE VIEW. IF THE PRESENTER'S INTERACTION WITH THE MODEL IS ALSO PERFORMED USING AN INTERFACE (E.G. IMODEL) THEN IT BECOMES DECOUPLED FROM BOTH OF ITS SIBLING COMPONENTS.
- UNFORTUNATELY, THE PRESENTER ILLUSTRATED IN THE EXAMPLE SEARCH APPLICATION DOES NOT PERFORM MUCH INTERMEDIARY BUSINESS LOGIC SUCH AS VALIDATION, EVENT HANDLING, LOGGING AND OTHER MIDDLE-TIER ACTIONS. HOWEVER, IT SHOULD BE CLEAR THAT THESE MIDDLE-TIER ACTIONS ARE IDEALLY SUITED TO THE PRESENTER AND CAN BE IMPLEMENTED AND TESTED RELATIVELY EASILY.

TESTING MVP APPLICATIONS
The Problem

- IN TRADITIONAL APPLICATIONS, THE USER INTERFACE PRESENTATIONAL TIER IS TIGHTLY COUPLED TO THE BUSINESS LOGIC TIER. THIS IS BECAUSE WIN AND WEB APPLICATIONS CONTAIN ‘CODE-BEHIND’ WHICH PERFORMS OPERATIONS BOTH DURING THE USER INTERFACE’S LIFECYCLE (E.G. PAGE LOAD) AND WHEN THE USER INTERACTS WITH THE UI (EVENT HANDLER ATTACHED TO A CONTROL’S EVENT E.G. BUTTON CLICK).
- THIS CODE-BEHIND BECOMES (PARTICULARLY IN LARGE OR COMPLEX APPLICATIONS) RESPONSIBLE FOR BOTH RENDERING AND BUSINESS LOGIC INVOCATION. THEREFORE THE CODE-BEHIND IS TIGHTLY COUPLED TO BOTH THE PRESENTATION TIER AND THE BUSINESS TIER. FURTHERMORE, IT IS FREQUENTLY NECESSARY FOR THE CODE-BEHIND TO INVOKE BUSINESS OPERATIONS AND THEN TRANSFORM THE RESULTS, HANDLE EXCEPTIONS AND PERFORM COMPLEX FLOW-CONTROL BASED ON THE RESULT OF AN OPERATION ETC. THE RESULTANT DESIGN CAN BECOME UNSTRUCTURED AND DIFFICULT TO MAINTAIN.
- A FURTHER, OFTEN NEGLECTED EFFECT OF BLOATED CODE-BEHIND IS AN UN-TESTABLE PRESENTATION TIER. ALTHOUGH THE APPEARANCE AND INTERACTION OF A USER INTERFACE CAN ONLY REALLY BE TESTED BY EITHER MANUAL OR SCRIPTED AUTOMATION TESTS, THE OVERUSE OF CODE-BEHIND MEANS THERE IS A GREAT DEAL OF LOGIC THAT HAPPENS IN THE EVENT HANDLERS IN THE CODE-BEHIND, BEFORE EXECUTION EVER PASSES TO THE WELL UNIT-TESTED BUSINESS TIER.

The Mvp Solution

- IMPLEMENTING THE MVP DESIGN PATTERN GREATLY INCREASES THE ABILITY TO TEST THE PRESENTATION TIER, SINCE THE VIEW COMPONENT CONSISTS ALMOST ENTIRELY OF VISUAL LOGIC (UI CONTROLS AND BASIC EVENT HANDLERS), AND ALL THE LOGIC SURROUNDING THE CALLS TO THE DATA TIERS (MODEL) IS ORCHESTRATED BY THE PRESENTER.
- THIS MEANS THAT IF THE TESTS CAN INVOKE PRESENTER FUNCTIONS AND INSPECT THE STATE OF THE VIEW, THE TESTS WILL COVER ALMOST ALL LOGIC CODE. THE ONLY PART OF THE APPLICATION LEFT UN-TESTED IS THE VISUAL STATE OF THE USER INTERFACE CONTROLS. DETERMINING THIS IS EXTREMELY DIFFICULT TO PERFORM PROGRAMMATICALLY AND AS SUCH IS NORMALLY LEFT TO MANUAL OR SCRIPTED AUTOMATION TESTS.
- THE BEST WAY OF TESTING AN MVP SOLUTION IS TO USE A ‘MOCK OBJECT’ IN PLACE OF THE ACTUAL VIEW. THE MOCK VIEW IMPLEMENTS THE IVIEW INTERFACE, AND IS ALSO CUSTOMISED TO ALLOW THE TESTS ACCESS TO VIEW MEMBERS WHICH ARE INACCESSIBLE IN THE ACTUAL VIEW.
- THIS TECHNIQUE OF MOCKING THE VIEW IS USED TO TEST THE EXAMPLE SEARCH APPLICATION, AS THE FOLLOWING CODE EXTRACT ILLUSTRATES.

```
public class ViewMock : IView

{
    public SearchPresenter Presenter { get; set; }

    public ViewMock()
    {
        this.Presenter = new Presenter(this);
    }

    #region ISearchView Members
    public List<string> Results { get; set; }
    public string SearchCriteria { get; set; }
    #endregion
}
```

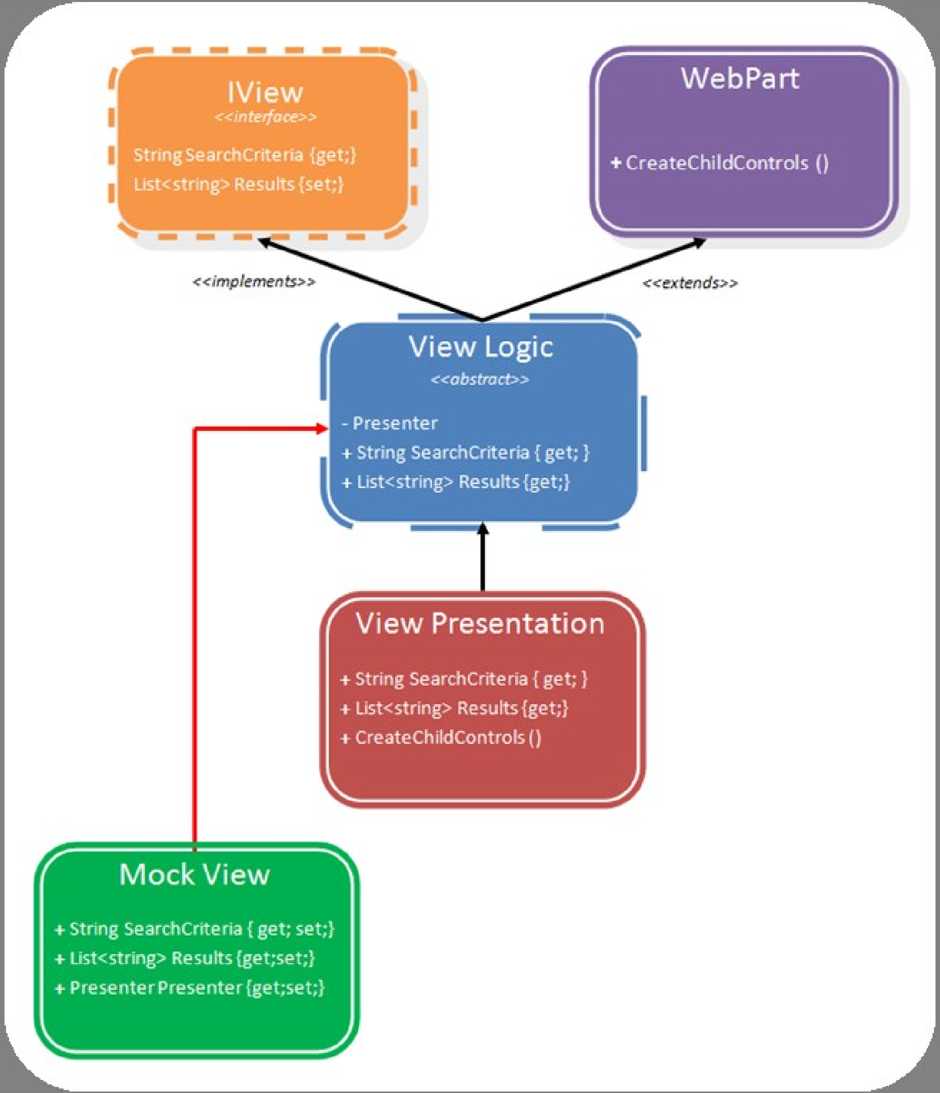
THE ABOVE CODE EXTRACT ILLUSTRATES THE MOCK VIEW IMPLEMENTATION CREATED TO TEST THE EXAMPLE SEARCH APPLICATION. IT IS WORTH NOTING HOWEVER THAT ALTHOUGH THE MOCK VIEW IMPLEMENTS THE IVIEW INTERFACE AS THE REAL WEB PART VIEW DOES, IT HAS AN EXTRA PROPERTY ‘PRESENTER’ AND BOTH SET AND GET PERMISSIONS ON ALL IMPLEMENTED IVIEW PROPERTIES (‘RESULTS’ AND ‘SEARCHCRITERIA’).

- THIS EXTRA ‘PRESENTER’ PROPERTY ALLOWS THE UNIT TEST TO OBTAIN THE PRESENTER INSTANCE WITHOUT RESORTING TO REFLECTION (IT IS NECESSARY TO OBTAIN THE VIEW’S PRESENTER INSTANCE, SINCE IT IS THE PRESENTER’S FUNCTIONS WHICH WILL BE TESTED).
- FURTHERMORE, THE GET AND SET ASSESSORS ON THE IVIEW IMPLEMENTED PROPERTIES ALLOW THE UNIT TEST TO INSPECT THE MOCK VIEW’S PROPERTY VALUES BEFORE AND AFTER A PRESENTER’S METHOD HAS BEEN INVOKED (SO AS TO ASSERT METHOD PRE AND POST CONDITIONS).
- WITHOUT TESTING THE MOCK VIEW AND INSTEAD TESTING THE ACTUAL VIEW, IT WOULD NOT BE POSSIBLE (WITHOUT USING COMPLEX AND UNRELIABLE REFLECTION) TO OBTAIN THE PRESENTER TO INVOKE BUSINESS OPERATIONS ON, SET-UP TEST SCENARIOS AND INSPECT THE VIEW’S STATE DURING TESTS.
- ONE CONCERN WHEN TESTING A MOCKED VIEW IS THAT THE EXTRA PROPERTIES AND INCREASED ACCESS TO THE VIEW MEMBERS IS THAT THE VIEW BEING TESTED NO LONGER RESEMBLES THE ACTUAL VIEW IN RESPECT OF DESIGN AND STRUCTURE. THIS CONCERN IS EASILY REMEDIED SINCE THE UNIT TESTS HOLDS REFERENCES TO BOTH THE MOCK VIEW INSTANCE DIRECTLY, AND ALSO VIA AN IVIEW REFERENCE (SEE FOLLOWING CODE EXTRACT).

```
/// <summary>
/// Search test. Results expected.
/// </summary>

[TestMethod()]
public void SearchTest_ResultsExpected()

{
    // Setup test pre-reqs and components
    ViewMock mockView = new ViewMock();
    Presenter presenter = mockView.Presenter;
    mockView.SearchCriteria = "test";
    ISearchView testView = mockView;
    Assert.IsTrue(testView.SearchCriteria.Equals(mockView.SearchCriteria));

    // Search
    presenter.Search();

    // Assert results were found
    // (min # results returned from search is 1 - error detail ietm)
    Assert.IsTrue(mockView.ResultsSet.Count > 0);

    // Assert results were valid, given the search criteria
    foreach (SearchResult r in mockView.ResultsSet)
    {
        Assert.IsNotNull(r);
        Assert.IsTrue(r.Name.IndexOf(testView.SearchCriteria) >= 0);
    }
}
```

A Further Mocking Example

- THE MOCKING SOLUTION ILLUSTRATED ABOVE MEETS THE REQUIREMENTS OF MOST SIMPLISTIC VIEWS THAT WILL BE DEVELOPED USING THE MVP PATTERN. HOWEVER AS DISCUSSED IN THE SECTION 'VIEW – SEPARATION OF CONCERNS' SOMETIMES IT IS NECESSARY TO IMPLEMENT A SMALL AMOUNT OF PRESENTATION TECHNOLOGY-SPECIFIC LOGIC TO PERFORM VALIDATION, CONVERSION OF USER INPUT TO BUSINESS TYPES AND VISA VERSA, ERROR HANDLING ETC.
- THE MOCK VIEW ILLUSTRATED PREVIOUSLY DOES NOT IMPLEMENT ANY OF THIS LOGIC AND WOULD REQUIRE EXTRA DEVELOPMENT TIME TO RE-IMPLEMENT THE VIEW LOGIC INSIDE THE MOCK VIEW. A WORKAROUND FOR THIS IS TO SPLIT THE VIEW INTO TWO TIERS; A LOGIC TIER AND A PRESENTATION TIER AND THEN CREATE THE MOCK VIEW AS A SUBCLASS OF THE ABSTRACT VIEW LOGIC TIER (SEE FOLLOWING DIAGRAM).


IView
<<interface>>
String SearchCriteria {get;}
List<string> Results {set;}
WebPart
+ CreateChildControls ()
<<implements>>
<<extends>>
View Logic
<<abstract>>
- Presenter
+ String SearchCriteria { get; }
+ List<string> Results {get;}
View Presentation
+ String SearchCriteria { get; }
+ List<string> Results {get;}
+ CreateChildControls ()
Mock View
+ String SearchCriteria { get; set;}
+ List<string> Results {get;set;}
+ Presenter Presenter {get;set;}

- THIS ALLOWS THE MOCK VIEW ACCESS TO THE PRESENTER INSTANCE, THE ABILITY TO EXPOSE NEW PROPERTIES AND MODIFY ASSESSORS TO ALLOW EASY INSPECTION OF VIEW STATE DURING TESTS AND MOST IMPORTANTLY THE ABILITY TO INVOKE ALL PRESENTATION TECHNOLOGY-SPECIFIC LOGIC HELD IN THE VIEW LOGIC TIER THROUGH INVOCATION OF THE MOCK VIEW'S BASE CLASS.
- THEREFORE, WHEN A UNIT TEST IS RUN WHICH INVOKES A METHOD ON THE PRESENTER, WHICH SETS A PROPERTY IN THE MOCK VIEW, THE OVERRIDDEN MOCK VIEW'S PROPERTY CAN INVOKE ITS BASE CLASS' PROPERTY WHICH IN TURN EXECUTES ANY VALIDATION ETC AGAINST THE INPUT (SEE FOLLOWING CODE FLOW DIAGRAM).

```
[TestMethod()]
public void SearchTest_ResultsExpected()
{
    // Mock View setup (hidden)...
    presenter.Search();
    // Assertions (hidden)...
    }
}

public class Presenter
{
    public void Search()
    {
            this.view.Results = Model.Search(this.view.SearchCriteria);
    }
    // Rest of Presenter class hidden...
}

public class MockViewLogic : ViewLogic
{
```

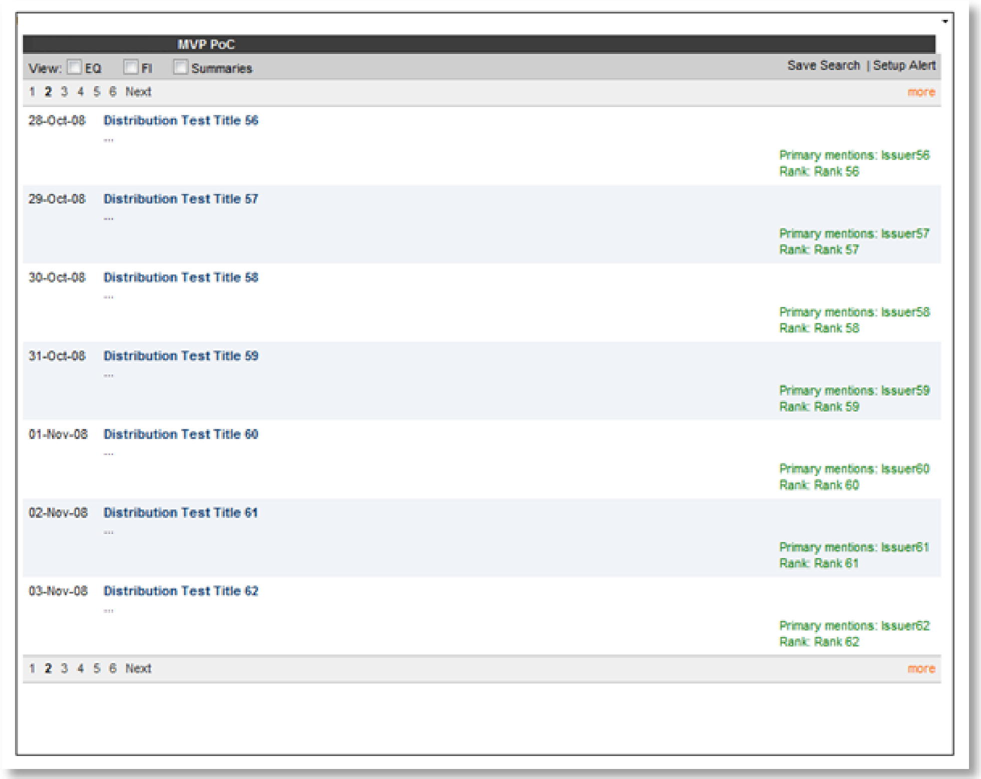
```
public override List<string> Results
    {
        set
        {
            base.Results = value;
        }
    }
    // Rest of mock view hidden...
}

public class ViewLogic: WebPart, IView
{
    public virtual List<string> Results
    {
        set
        {
            // Validation
            // Error handling
            // Localisation
            // etc...
        }
    }
    // Rest of view logic abstract class hidden...
}
```

THIS FORM OF MOCKING THE MVP VIEW ALLOWS FOR MORE PERVASIVE AND COMPLETE TESTING OF THE VIEW COMPONENT.

Presentational Interoperability

THE FOLLOWING CODE EXTRACTS ARE GIVEN SO AS TO ILLUSTRATE HOW TO IMPLEMENT A VIEW FOR THE EXAMPLE SEARCH APPLICATION, GIVEN ITS SIMPLE IVIEW INTERFACE. ONE VIEW IS IMPLEMENTED AS AN ASP.NET WEB PART, THE OTHER AS A CLIENT SIDE WPF APPLICATION.

ASP.NET Web Part Implementation



MVP PoC

View: ☐ EQ ☐ FI ☐ Summaries

Save Search | Setup Alert

1 **2** 3 4 5 6 Next — more

28-Oct-08 **Distribution Test Title 56**
...
Primary mentions: Issuer56
Rank: Rank 56

29-Oct-08 **Distribution Test Title 57**
...
Primary mentions: Issuer57
Rank: Rank 57

30-Oct-08 **Distribution Test Title 58**
...
Primary mentions: Issuer58
Rank: Rank 58

31-Oct-08 **Distribution Test Title 59**
...
Primary mentions: Issuer59
Rank: Rank 59

01-Nov-08 **Distribution Test Title 60**
...
Primary mentions: Issuer60
Rank: Rank 60

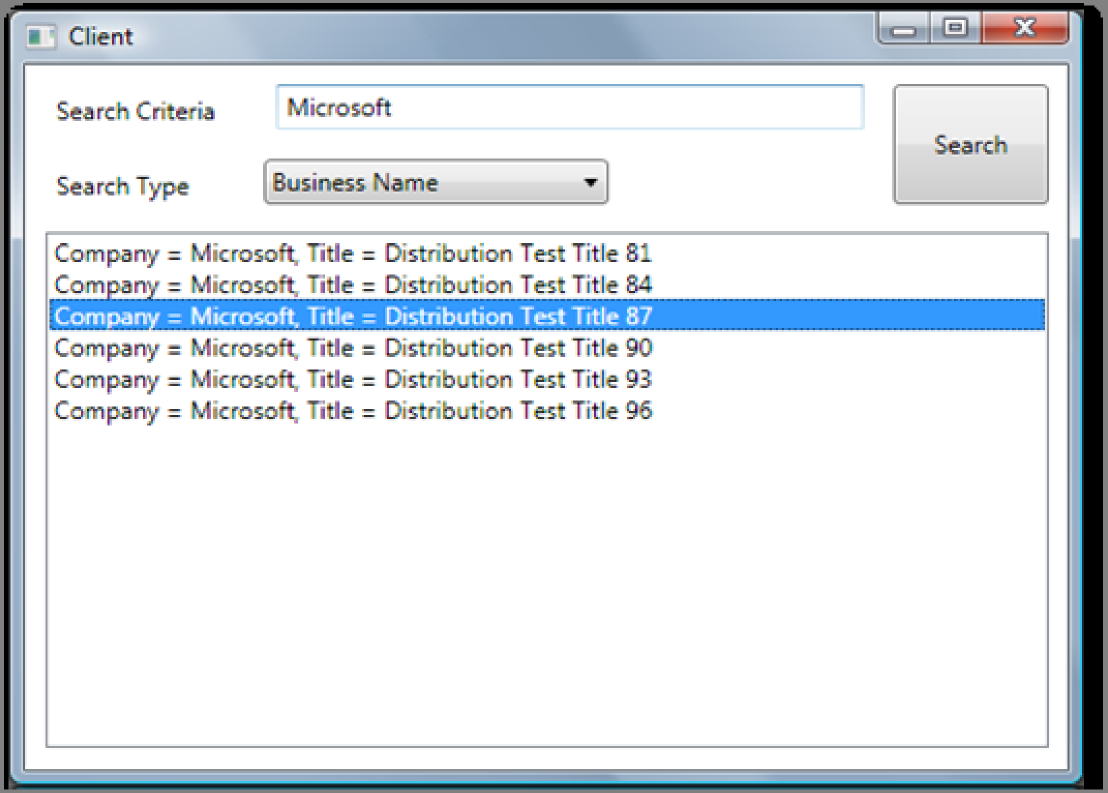
02-Nov-08 **Distribution Test Title 61**
...
Primary mentions: Issuer61
Rank: Rank 61

03-Nov-08 **Distribution Test Title 62**
...
Primary mentions: Issuer62
Rank: Rank 62

1 **2** 3 4 5 6 Next — more

```
public class ViewImpl : WebPart, IView
    {
        private Presenter presenter;
        Label lblSearch = new Label() { Text = "Enter bank to search for: " };
        Button btnSearch = new Button() { Text = "Search" };
        BulletedList resultsList = new BulletedList();
        TextBox tbSearchCriteria = new TextBox();

        public ViewImpl()
        {
            this.presenter = new Presenter(this);
        }

        protected override void CreateChildControls()
        {
            this.Controls.Clear();
            this.btnSearch.Click += new EventHandler(btnSearch_Click);
            this.Controls.Add(this.lblSearch);
            this.Controls.Add(this.tbSearchCriteria);
            this.Controls.Add(this.btnSearch);
            this.Controls.Add(this.resultsList);
            this.ChildControlsCreated = true;
        }
    void btnSearch_Click(object sender, EventArgs e)
    {
        this.presenter.Search();
    }

    #region IView Members
    public string SearchCriteria
    {
        get { return this.tbSearchCriteria.Text; }
    }

    public List<string> Results
    {
        set
        {
            this.resultsList.Items.Clear();
            List<string> results = new List<string>(value);
            foreach (string r in results)
            {
                this.resultsList.Items.Add(r);
            }
        }
    }
    #endregion
}
```

WPF Implementation



Microsoft

Business Name

```
Company = Microsoft, Title = Distribution Test Title 81
Company = Microsoft, Title = Distribution Test Title 84
Company = Microsoft, Title = Distribution Test Title 87
Company = Microsoft, Title = Distribution Test Title 90
Company = Microsoft, Title = Distribution Test Title 93
Company = Microsoft, Title = Distribution Test Title 96
```

```
public partial class Client : Window, IView
{
    SearchPresenter presenter;
    public Client()
    {
        this.presenter = new SearchPresenter(this);
        InitializeComponent();
    }

    private void button1_Click(object sender,
RoutedEventArgs e)
    {
        this.presenter.Search();
    }

    #region ISearchView Members
public List<string> Results
    {
        set
        {
            this.listBox1.Items.Clear();
            foreach (SearchResult r in value)
            {
                this.listBox1.Items.Add(r.ToString());
            }
        }
    }

    public string SearchCriteria
    {
        get { return this.tbSearchCriteria.Text; }
    }
    #endregion
}
```

Links

- HTTP://BLOG.VUSCODE.COM/MALOVICN/ARCHIVE/2007/11/04/MODEL-VIEW-PRESENTER-MVP-DESIGN-PATTERN-CLOSE-LOOK-PART-2-PASSIVE-VIEW.ASPX
- HTTP://WEBLOGS.ASP.NET/BSIMSER/ARCHIVE/2006/07/18/MODEL_2D00_VIEW_2D00_PRESENTER-PATTERN-WITH-SHAREPOINT-WEB-PARTS.ASPX
- HTTP://GEEKSWITHBLOGS.NET/PODWYSOCKI/ARCHIVE/2007/12/20/117881.ASPX
- HTTP://EN.WIKIPEDIA.ORG/WIKI/MODEL_VIEW_PRESENTER
- HTTP://EN.WIKIPEDIA.ORG/WIKI/MOCK_OBJECT
- HTTP://MSDN.MICROSOFT.COM/EN-US/MAGAZINE/CC188690.ASPX
- HTTP://WWW.MARTINFOWLER.COM/EAADEV/MODELVIEWPRESENTER.HTML
- HTTP://EN.WIKIPEDIA.ORG/WIKI/FACTORY_METHOD