CSC320-A

Module 4

Implementing a Windows Forms Application Framework (Part 2)

1 Workspaces & Views

1.1 Abstraction

Just like an action site representing an insertion point for commands, a workspace is insertion point for views. Workspaces and views are user-interface elements for users to interact with the application. While workspaces are predefined during design-time, views are dynamically constructed during runtime. Modules, services and commands can create views and attached them to workspaces we exposed through the shell.

Unlike command that requires only a single implementation, each view is different so abstraction is required. We will first add an interface named **IView** to contain a basic set of properties accessible by a workspace. Add an **IWorkspace** interface that has methods where views can be added, removed and updated.

Interface for views: Symbion\IView.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Symbion {
    public interface IView {
        Image Icon { get; }
        string Caption { get; }
        Control Control { get; }
        IWorkspace Workspace { get; set; }
        void Show(string workspaceId);
        void Close();
    }
}
```

Interface for workspaces: Symbion\IWorkspace.cs

```
using System;
namespace Symbion {
    public interface IWorkspace {
        void Append(IView view);
        void Remove(IView view);
        void Update(IView view);
    }
}
```

1.2 Base Implementation

We can provide a base implementation for the abstractions. Visual Studio provides design-time support for **Form** and **UserControl** classes so these are good choices to use as base classes. Since a view is not suppose to be a window, **UserControl** class. However a workspace is free to create a window to host each view.

While there are some basic code that can be inherited from a **BaseView** class, every workspace implementation would be totally different so it's completely abstract like an interface but still having a base class allows you to easily add shared members in the future should you need to. For example we can also expose an **IShell** singleton in the classes so that they can easily access UI services. Note these base classes cannot be abstract for inherited control support.

Base view class: BaseView.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Symbion {
    public class BaseView : UserControl, IView {
        public virtual Image Icon { get; set; }
        public virtual string Caption { get; set; } = string.Empty;
        public Control Control { get { return this; } }
        public IWorkspace Workspace { get; set; }
        private static IShell _shell;
        public static IShell Shell {
            get { return _shell ?? (_shell = ServiceRepository.Get<IShell>()); }
        }
        public void Show(string workspaceId) {
            Shell.Workspaces[workspaceId].Append(this);
        }
        public void Close() { if (Workspace != null) Workspace.Remove(this); }
    }
}
```

Base workspace class: BaseWorkspace.cs

```
namespace Symbion {
    public abstract class BaseWorkspace : UserControl, IWorkspace {
        public virtual void Append(IView view) { }
        public virtual void Remove(IView view) { }
        public virtual void Update(IView view) { }

        private static IShell _shell;
        public static IShell Shell {
            get { return _shell ?? (_shell = ServiceRepository.Get<IShell>()); }
        }
    }
}
```

1.3 Exposing Workspaces

You can update the **IShell** interface to expose application workspaces. In **SymBank** application, update **ShellForm** class to provide a dictionary to register workspaces to be accessible through **IShell**.

Exposing workspaces in Shell: Symbion\IShell.cs

```
public interface IShell : IService {
    Dictionary<string, IActionSite> Sites { get; }
    Dictionary<string, IWorkspace> Workspaces { get; }
                :
}
```

Implementing workspace support in application: SymBank\ShellForm.cs

```
private Dictionary<string, IActionSite> _sites;
private Dictionary<string, IWorkspace> _workspaces;

public ShellForm() {
    InitializeComponent();
    _sites = new Dictionary<string, IActionSite>();
    _workspaces = new Dictionary<string, IWorkspace>();
                :
}

public Dictionary<string, IWorkspace> Workspaces {
    get { return _workspaces; }
}
```

1.4 Implementing a Workspace

We will now implement a type of workspace. Add a class named as **TabWorkspace** in **Symbion** using the **Inherited User Control** template. You will then be prompted to select the base user control class. Select the **BaseWorkspace** class and click OK to confirm. We can then visually design the workspace. Add and dock a **ToolStrip** to the top of the control named **tbrMain**. Add and dock a **TabControl** in the center named **tabMain** and remove all the tab pages. We will use a dictionary to associate a view to each **TabPage** which will host the view.

Implementing a TabControl workspace: TabWorkspace.cs

```
namespace Symbion {
    public partial class TabWorkspace : BaseWorkspace {
        private Dictionary<IView, TabPage> _views;
        public TabWorkspace() {
            InitializeComponent();
            _views = new Dictionary<IView, TabPage>();
        }
    }
}
```

Adding a view to workspace

```
public override void Append(IView view) {
    TabPage page = new TabPage(view.Caption);
    view.Control.Dock = DockStyle.Fill;
    page.Controls.Add(view.Control);
    tabMain.TabPages.Add(page);
    _views.Add(view, page);
    view.Workspace = this;
}
```

Removing view from workspace

```
public override void Remove(IView view) {
    TabPage page = _views[view];
    tabMain.TabPages.Remove(page);
    _views.Remove(view);
    view.Workspace = null;
}
```

Updating view host in workspace

```
public override void Update(IView view) {
    TabPage page = _views[view];
    page.Text = view.Caption;
}
```

We can now add this workspace into our application. Go to the **SymBank** project and open designer view for **ShellForm**. Look in *Toolbox* window for the **TabWorkspace** control. If you cannot see it that means that you did not rebuild **Symbion** project. First add a **SplitContainer** in the form named **spcMain** and add the **TabWorkspace** into the second panel named **tabWorkspace**. You can now register the workspace as shown below.

Registering the workspace: ShellForm.cs

```
public ShellForm() {
    InitializeComponent();
    _sites = new Dictionary<string, IActionSite>();
    _workspaces = new Dictionary<string, IWorkspace>();
    _workspaces.Add("TabSpace", tabWorkspace);
            :
}
```

1.5 Implementing a View

We will now implement a view. Views are normally implemented by the application or modules. We can still implement common views in **Symbion** like for example a web browser view. Add an Inherited User Control named as **WebBrowserView** extended from **BaseView** class.

Open the class in designer view to visually implement the view. Set the **BorderStyle** property of the view to **Fixed3D**. Add a **ToolStrip** to the top named **tbrMain**. Create 2 buttons in **ToolStrip** named **btnBack**, **btnForward** followed by a **TextBox** named **txtURL** and followed by two buttons named **btnNavigate** and **btnClose**. Finally add and dock a **WebBrowser** in the center of the control names **webBrowser**. Assign the icons provided to each button. Create **Click** event handlers for each button and **DocumentCompleted** event for the **WebBrowser** control. In the code add an **Open** method that can be called to use the web browser control to navigate to a URL passed in as a parameter.

Basic class content for view: Symbion\WebBrowserView.cs

```
using System;
using System.Windows.Forms;

namespace Symbion {
    public partial class WebBrowserView : BaseView {
        public WebBrowserView() {
            InitializeComponent();
        }
    }
}
```

Expose method to be callable from code

```
public void Open(string url) {
    try {
        txtURL.Text = url;
        webBrowser.Navigate(url);
    }
    catch { }
}
```

Event handlers for buttons

```
private void btnBack_Click(object sender, EventArgs e) {
    if (webBrowser.CanGoBack) webBrowser.GoBack();
}

private void btnForward_Click(object sender, EventArgs e) {
    if (webBrowser.CanGoForward) webBrowser.GoForward();
}

private void btnNavigate_Click(object sender, EventArgs e) {
    var text = txtURL.Text.Trim();
    if (text.Length > 0) Open(text);
}

private void btnClose_Click(object sender, EventArgs e) {
    Close();
}
```

Event handler for web browser

```
private void webBrowser_DocumentCompleted(
    object sender, WebBrowserDocumentCompletedEventArgs e) {
            Caption = webBrowser.DocumentTitle;
            Workspace.Update(this); // make sure view host is updated
        }
    }
}
```

1.6 Implementing a Service

Because our view is implemented in **Symbion**, it can be used directly by all modules and services. You can instantiate the view directly or use a **Command**. However since we will isolate modules from each other, how would is be possible for one module to create a view implemented in another module? This can be done by implementing a service to create the views. The following shows a service interface that exposes an Open method that when called will automatically create a view and then add it to the workspace identified by the ID.

WebBrowserService interface: IWBrowserService.cs

```
namespace Symbion {
    public interface IWebBrowserService : IService {
        void Open(string url, string workspaceId);
    }
}
```

WebBrowserService implementation: WebBrowserService.cs

```
using System;

namespace Symbion {
    public class WebBrowserService : BaseService, IWebBrowserService {
        public void Open(string url, string workspaceId) {
            WebBrowserView view = new WebBrowserView();
            view.Open(url); view.Show(workspaceId);
        }
    }
}
```

You can then register the service and access it from anywhere within the application or external modules and services. You can directly launch the **WebBrowserView** on startup or create a **Command** to assign to a menu item or toolbar button to activate the same code.

Registering and using WebBrowserService: SymBank\ShellForm.cs

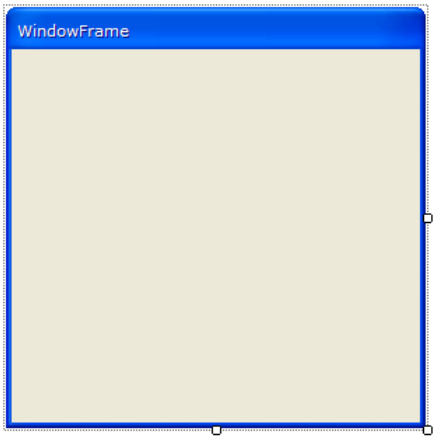
```
private void ShellForm_Load(object sender, EventArgs e) {
        :
    WebBrowserService wbs = new WebBrowserService();
    wbs.Add<IWebBrowserService>();
    wbs.Open("http://www.microsoft.com","TabSpace");
}
```

2 Additional Workspaces

2.1 Window Workspace

The functionality of the window workspace is to open each view in a separate window. This would be useful for views displaying input forms, reports and messages. We will first begin by creating a separate form class that is used for hosting the view. Since it is mainly to provide a window frame for the view, we will remove all other features of the form.

Visual appearance of form




```
AutoScroll        : true
Controlbox        : false
ShowInTaskBar     : false
StartPosition     : CenterScreen
Title             : WindowFrame
```

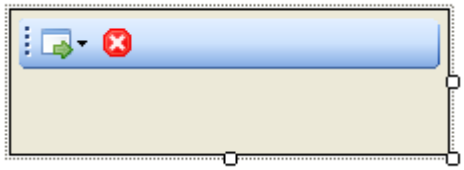
The **WindowFrame** class will also help to store information which can be used to access the workspace, view and the component on the workspace a user can select to activate the window. The workspace will still appear in the shell so that the user can use it to select and manage the windows.

Once that you have implemented the **WindowFrame** class we can then create the workspace. Extend **WindowWorkspace** from **BaseWorkspace**. Set **BorderStyle** to **Fixed3D** and set **Padding** to **4**. Add a **Toolstrip** to the top named **tbrMain** that will contain a **ToolStripDownDownButton** followed by a **ToolStripButton** named **btnWindowList** and **btnCloseAll** respectively. Use the following as button images. For each view opened, a menu item will be added to the drop down that a user can select to activate the window.

Icon images used

application\application_go.png
dialog\cancel.png

Visual appearance of workspace



Window frame implementation: Symbion\WindowFrame.cs

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Symbion {
    public partial class WindowFrame : Form {
        protected IWorkspace workspace;
        protected Component item;
        protected IView view;

        public IView View { get { return view; }}
        public Component Item { get { return item; }}

        public WindowFrame(
            IWorkspace workspace,
            Component item,
            IView view) {
            InitializeComponent();
            this.workspace = workspace;
            this.item = item;
            this.view = view;
            Text = view.Caption;
            Control viewControl = view.Control;
            viewControl.Dock = DockStyle.Fill;
            ClientSize = viewControl.Size;
            Controls.Add(viewControl);
        }
    }
}
```

Window workspace implementation: WindowWorkspace.cs

```
namespace Symbion {
    public partial class WindowWorkspace : BaseWorkspace {
        protected Dictionary<IView, WindowFrame> _views;

        public WindowWorkspace() {
            InitializeComponent();
            _views = new Dictionary<IView, WindowFrame>();
            int height = tbrMain.Height + Padding.Top + Padding.Bottom;
            MinimumSize = new Size(0, height);
        }
    }
}
```

Activate window when menu item is selected

```
protected void OnItemClick(object sender, EventArgs e) {
    ToolStripItem item = (ToolStripItem)sender;
    WindowFrame form = (WindowFrame)item.Tag;
    form.Activate();
}
```

Remove all views and close all windows

```
private void btnCloseAll_Click(object sender, EventArgs e) {
    foreach (IView view in _views.Keys) Remove(view);
}
```

Add menu item and window for each view added

```
public override void Append(IView view) {
    ToolStripItem listItem = btnWindowList.DropDownItems.Add(view.Caption);
    WindowFrame frame = new WindowFrame(this, listItem, view);
    listItem.Click += OnItemClick;
    listItem.Tag = frame;
    frame.Show();
    _views.Add(view, frame);
    view.Workspace = this;
}
```

Remove menu item and close window for the view

```
public override void Remove(IView view) {
    WindowFrame frame = _views[view];
    btnWindowList.DropDownItems.Remove((ToolStripItem)frame.Item);
    frame.Close(); _views.Remove(view);
    view.Workspace = null;
}
```

Update menu item and view caption

```
public override void Update(IView view) {
    WindowFrame frame = _views[view];
    ToolStripItem listItem = (ToolStripItem)frame.Item;
    listItem.Text = frame.Text = view.Caption;
}
```

2.2 Deck Workspace

Another common workspace is a vertical list of bars where each bar represents one view. Only the top bar will show the contents of the view. Selecting any bar will make it the top bar and the previous top bar will move to the bottom of the deck. We will be using existing Windows Forms controls to create this kind of behavior. To simulate a bar on the deck workspace, we will use a button but since there are a quite a number of settings required to make the button look good in the workspace, we are going to extend our own button class. Add a new class named **DeckButton** in and extend it from the **Button** class. In the constructor, we will customize the button to our liking and to also provide one that can accept an **IView** instance.

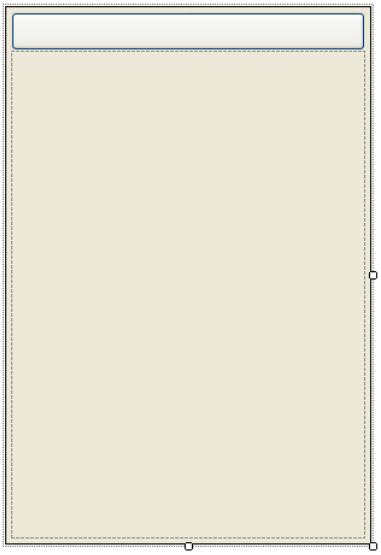
Custom button class: DeckButton.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Symbion {
    public class DeckButton : Button {
        public DeckButton() {
            Height = 32;
            Padding = new Padding(4, 0, 0, 0);
            ImageAlign = ContentAlignment.MiddleLeft;
            TextAlign = ContentAlignment.MiddleLeft;
            TextImageRelation = TextImageRelation.ImageBeforeText;
            Dock = DockStyle.Bottom;
        }
        public DeckButton(IView view) : this() {
            Image = view.Icon;
            Text = view.Caption;
            Tag = view;
        }
    }
}
```

We can now add in a **DeckWorkspace** class extended from **BaseWorkspace**. As usual, set the **BorderStyle** to **Fixed3D** and **Padding** to **4**. In the workspace add a **DeckButton** instance named as **btnActive** and dock it to the top. Then add a **Panel** named as **viewPanel** docked to the center to be used to contain the current active view. New deck buttons added will be docked to the bottom of the workspace. Create a **Click** event handler for **btnActive**.

Visual appearance of view



To remember the ordinal order of the buttons, we will need to have an extra **List** collection named **_queue** since a **Dictionary** is not ordered according to which item was added first. The first item is assigned to the **activeButton** field where its content is copied to the **btnActive** button. When a new button is selected the previous **activeButton** is added back into the queue and docked to the bottom and the new button is then removed from the queue and assign to **activeButton** field. Clicking on **btnActive** would make the button at the top of the queue as the active button.

Workspace implementation: Deckworkspace.cs

```
namespace Symbion {
    public partial class DeckWorkspace : BaseWorkspace {
        protected Dictionary<IView, DeckButton> _views;
        protected List<DeckButton> _queue;
        protected DeckButton _activeButton;
```

```
        public DeckWorkspace() {
            InitializeComponent();
            _views = new Dictionary<IView, DeckButton>();
            _queue = new List<DeckButton>();
        }
    }
}
```

Configuring active button from selected button

```
protected void SetActiveButton(DeckButton item) {
    IView view = (IView)item.Tag;
    view.Control.Dock = DockStyle.Fill;
    viewPanel.Controls.Add(view.Control);
    btnActive.Text = view.Caption;
    btnActive.Image = view.Icon;
    btnActive.Enabled = true;
    _activeButton = item;
}
```

Processing selection of a button

```
protected void ActivateNewButton(DeckButton item) {
    if (_activeButton != null) {
        IView view = (IView)_activeButton.Tag;
        viewPanel.Controls.Remove(view.Control);
        Controls.Add(_activeButton);
        _queue.Add(_activeButton);
    }
    _queue.Remove(item);
    Controls.Remove(item);
    SetActiveButton(item);
}
```

Detect selection of button on click event

```
protected void OnItemClick(object sender, EventArgs e) {
    ActivateNewButton((DeckButton)sender);
}
```

Toggle views when active button is clicked

```
private void btnActive_Click(object sender, EventArgs e) {
    if (_queue.Count > 0) ActivateNewButton(_queue[0]);
}
```

Adding a button for each view

```
public override void Append(IView view) {
    DeckButton item = new DeckButton(view);
    item.Click += OnItemClick;
    if (_views.Count == 0) SetActiveButton(item); else {
        _queue.Add(item); Controls.Add(item);
    }
    _views.Add(view, item);
    view.Workspace = this;
}
```

Removing button together with view

```
public override void Remove(IView view) {
    DeckButton item = _views[view];
    if (item == _activeButton) {
        _activeButton = null;
        viewPanel.Controls.Remove(view.Control);
        if (_queue.Count == 0) {
            btnActive.Image = null;
            btnActive.Text = string.Empty;
            btnActive.Enabled = false;
        }
        else ActivateNewButton(_queue[0]);
    }
    else {
        Controls.Remove(item);
        _queue.Remove(item);
    }
    _views.Remove(view);
    view.Workspace = null;
}
```

Updating view host

```
public override void Update(IView view) {
    DeckButton item = _views[view];
    item.Image = view.Icon;
    item.Text = view.Caption;
    if (_activeButton.Tag == view)
        SetActiveButton(item);
}
```

2.3 Registering Workspaces

We can now return back to **SymBank** application can register all the new workspaces as well. Use the web browser view to test the different available workspaces. To test **DeckWorkspace** you will need to add multiple views.

Register workspaces: ShellForm.cs

```
public ShellForm() {
    InitializeComponent();
    _sites = new Dictionary<string, IActionSite>();
    _workspaces = new Dictionary<string, IWorkspace>();
    _workspaces.Add("TabSpace", tabWorkspace);
    _workspaces.Add("WindowSpace", windowWorkspace);
    _workspaces.Add("DeckSpace", deckWorkspace);
                :
    }
}
```