SSE 661

Software Architecture

Project #2

by

Jason Payne

June 10, 2015

TABLE OF CONTENTS

Topics Covered	Topic Examples
System Design Patterns	• Microkernel • Wrapper-Facade

1. Prototype Application

For the sake of this project, a custom application named '***My Favorite Homes***' will be used to exercise the different black box testing techniques against. This application follows the idea that a searching home buyer will need an application to help them track the properties they are most interested in. In real estate, there are two main types of homes that are offered for sale. Those that are listed in the Multiple Listing Service (MLS) with real estate agents and those that are 'For Sale By Owner' (FSBO) and not listed in the MLS. With the prevalence of internet tools made available to sellers, more and more homes are being listed without real estate agents and/or not in the MLS. This is what makes this application a viable solution for potential home buyers. This allows potential buyers a tool that allows them to track interesting properties without worrying about what website or MLS the home is listed at.

1.1 Requirements

This application is a simple GUI that is presented to the user upon launching the executable (see Figure 2). From here, the user can enter the address (street, city, state, zip code), number of bedrooms, number of bathrooms, size in sq. ft., the year the home was built, and the price. Once valid input has been entered, the information is stored and displayed in a list-like fashion (see Figure 3). If an item needs to be removed from the list, the user is able to easily remove single or multiple items through the main GUI (see Figure 4).

With a list of favorite homes created, the user has a few options: 1) to save the file to a specially formatted binary (**F**avorite **H**omes **P**ositions, *.fhp) file that is intended for use by GPS devices (see Figure 5), and/or 2) to export the list to a spreadsheet for the purposes of sharing or printing the list of favorite homes and their respective data (see Figure 6). Ideally, these features would be used in conjunction so that the home buyers could create their list, upload it to a supported GPS device, and share the exported results with others for printing hard copies or sending to a mobile device.

1.2 Source Code

The intent of this solution was to provide an early prototype. This is an important step because often times a prototype version can secure contracts and clarify questions or any concerns over requirements. With that said, there was no detailed analysis of the design and architecture of the system. All of the source code was placed into a single executable project and split amongst two forms (WinForms) and a main program. That source code is presented below.

SaveFhpFileDialog

This class contains the business logic used when prompting the user for the name of the *.fhp file where the binary data will be saved. The Save and Cancel buttons are designed to return `DialogResults` so that the calling code will know if the user has completed the operation. The

`Filename` property simply provides a way for calling code to retrieve the name of the file that the user has specified. The designer view is shown in Figure 1.

```
using System.Windows.Forms;

namespace Baseline
{
    public partial class SaveFhpFileDialog : Form
    {
        public SaveFhpFileDialog()
        {
            InitializeComponent();
        }

        /// <summary>
        /// The name of the binary *.fhp file where the data will be stored.
        /// </summary>
        public string Filename
        {
            get { return textBoxFilename.Text + ".fhp"; }
        }
    }
}
```

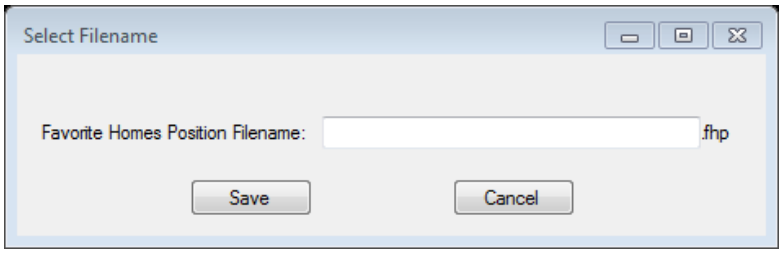

Figure 1: *Designer view of the* ***`SaveFhpFileDialog`*** *form*

FormFavoriteHomes

This class contains most of the business logic for the main features of the '*My Favorite Homes*' application. Figure 2 shows the designer view of the form. The highlighted items indicate dependencies that would make scalability and portability very limited, if not impossible. The Microkernel and Wrapper-Façade design patterns will help address these design "forces" so that the application is not tightly coupled to specific file systems or specific application libraries.

```
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Excel=Microsoft.Office.Interop.Excel;

namespace Baseline
{
    public partial class FormFavoriteHomes : Form
    {
        public FormFavoriteHomes()
        {
            InitializeComponent();
        }
```

```
        /// <summary>
        /// Closes the application when the Close button is pressed.
        /// </summary>
        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }

        /// <summary>
        /// Adds new homes to the list when the Add button is pressed.
        /// </summary>
        private void buttonAdd_Click(object sender, EventArgs e)
        {
            var price = Double.Parse(textBoxPrice.Text.Remove(0, 1));
            var size = Int32.Parse(textBoxSize.Text);
            var pricePerSqFt = ((price == 0.0) || (size == 0)) ? 0.0 : price / size;

            favoriteHomesDataset.dt.Rows.Add(
                textBoxStreet.Text + ", " +
                textBoxCity.Text + ", " + maskedBoxState.Text + ", " + maskedBoxZip.Text,
                Int32.Parse(textBoxBedrooms.Text),
                Int32.Parse(textBoxBathrooms.Text),
                size,
                Int32.Parse(textBoxYearBuilt.Text),
                textBoxPrice.Text,
                pricePerSqFt.ToString("C") + " / sq. ft.");
        }

        /// <summary>
        /// Deletes homes from the list when the Delete button is pressed.
        /// </summary>
        private void buttonDelete_Click(object sender, EventArgs e)
        {
            var selectedRows = dataGridView.SelectedRows;
            foreach (DataGridViewRow row in selectedRows)
            {
                dataGridView.Rows.Remove(row);
            }

        }

        /// <summary>
        /// Listener for determining if the Delete button should be enabled or not.
        /// </summary>
        private void dataGridView_RowStateChanged(object sender,
                                                  DataGridViewRowStateChangedEventArgs e)
        {
            // If the list is not empty and the Delete button is disabled, then enable the
            // Delete button...
            if ((dataGridView.SelectedRows.Count > 0) &&
                (!buttonDelete.Enabled))
            {
                buttonDelete.Enabled = true;
            }
            // Else, if the list is empty and the Delete button is enabled, then disabled the
            // Delete button.
            else if ((dataGridView.SelectedRows.Count == 0) &&
                     (buttonDelete.Enabled))
            {
                buttonDelete.Enabled = false;
            }
```

```
        }

        /// <summary>
        /// Initiates the process that allows the user to save the list to a *.fhp file.
        /// </summary>
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var dlg = new SaveFhpFileDialog();

            if (dlg.ShowDialog() == DialogResult.OK)
            {
                var recordSize = 26 + // Street (26 characters)
                                 2 + // formatting
                                 26 + // City (26 characters)
                                 2 + // formatting
                                 2 + // State (2 characters)
                                 2 + // formatting
                                 5 + // Zip (5 characters)
                                 1 + // Bedrooms (1 character)
                                 1 + // Bathrooms (1 character)
                                 4 + // Size (4 characters)
                                 4 + // Year Built (4 characters)
                                 7 // Price (7 characters)
                    ;
                var homes = favoriteHomesDataset.dt;
                var buffer = new byte[1 + (recordSize*homes.Count)];
                buffer[0] = Convert.ToByte(homes.Count);
                var filename = String.Format(@"{0}\{1}\{2}",
                                             Environment.GetFolderPath(
                                                 Environment.SpecialFolder
                                                            .CommonApplicationData),
                                             "My Favorite Homes", dlg.Filename);
                var fileInfo = new FileInfo(filename);

                if (!Directory.Exists(fileInfo.DirectoryName))
                    Directory.CreateDirectory(fileInfo.DirectoryName);

                var index = 1;

                for (var i = 0; i < homes.Count; i++)
                {
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Address), 0,
                                     buffer, index, homes[0].Address.Length);
                    index += 65;
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Bedrooms.ToString()), 0,
                                     buffer, index, 1);

                    index += 1;
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Bathrooms.ToString()), 0,
                                     buffer, index, 1);

                    index += 1;
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Size.ToString()), 0,
                                     buffer, index, 4);

                    index += 4;
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Year_Built.ToString()),
                                     0, buffer, index, 4);

                    index += 4;
                    Buffer.BlockCopy(Encoding.ASCII.GetBytes(homes[0].Price), 0,
```

```
                                    buffer, index, homes[0].Price.Length);

                    index += 7;
                }
                File.WriteAllBytes(filename, buffer);
                MessageBox.Show("File succesfully saved!");
            }
        }

        /// <summary>
        /// Initiates the process that allows the user to export the list to a spreadsheet.
        /// </summary>
        private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var favoriteHomes = favoriteHomesDataset.dt;

            if (!favoriteHomes.Any()) return;

            var app = new Excel.Application {Visible = true, SheetsInNewWorkbook = 1};
            app.Workbooks.Add();

            var worksheet = (Excel.Worksheet)app.ActiveSheet;
            var headerRow = (Excel.Range)worksheet.Rows[1];
            headerRow.Font.Bold = true;

            worksheet.Cells[1, 1] = "Address";
            worksheet.Cells[1, 2] = "Bedrooms";
            worksheet.Cells[1, 3] = "Bathrooms";
            worksheet.Cells[1, 4] = "Size";
            worksheet.Cells[1, 5] = "Year Built";
            worksheet.Cells[1, 6] = "Price";
            worksheet.Cells[1, 7] = @"$ / sq. ft.";

            for (int i = 0; i < favoriteHomes.Count; i++)
            {
                var row = i + 2;
                var home = favoriteHomes[i];

                worksheet.Cells[row, 1] = home.Address;
                worksheet.Cells[row, 2] = home.Bedrooms;
                worksheet.Cells[row, 3] = home.Bathrooms;
                worksheet.Cells[row, 4] = home.Size;
                worksheet.Cells[row, 5] = home.Year_Built;
                worksheet.Cells[row, 6] = home.Price;
                worksheet.Cells[row, 7] = home.PricePerSqFt;
            }

            worksheet.Columns.AutoFit();

            app.Quit();
        }

        /// <summary>
        /// Closes the application when the user selects the 'File > Exit' menu option.
        /// </summary>
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        /// <summary>
```

```
        /// Initiates the process of clearing all of the items from the list by first
        /// getting confirmation from the user and then actually clearing the items from the
        /// list if the user confirmed the clear all action.
        /// </summary>
        private void clearAllToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var result =
                MessageBox.Show("Are you sure you wish to clear all items from the list?",
                                "Clear All Items?", MessageBoxButtons.YesNo);
            if(result.Equals(DialogResult.Yes))
                favoriteHomesDataset.Clear();
        }
    }
}
```

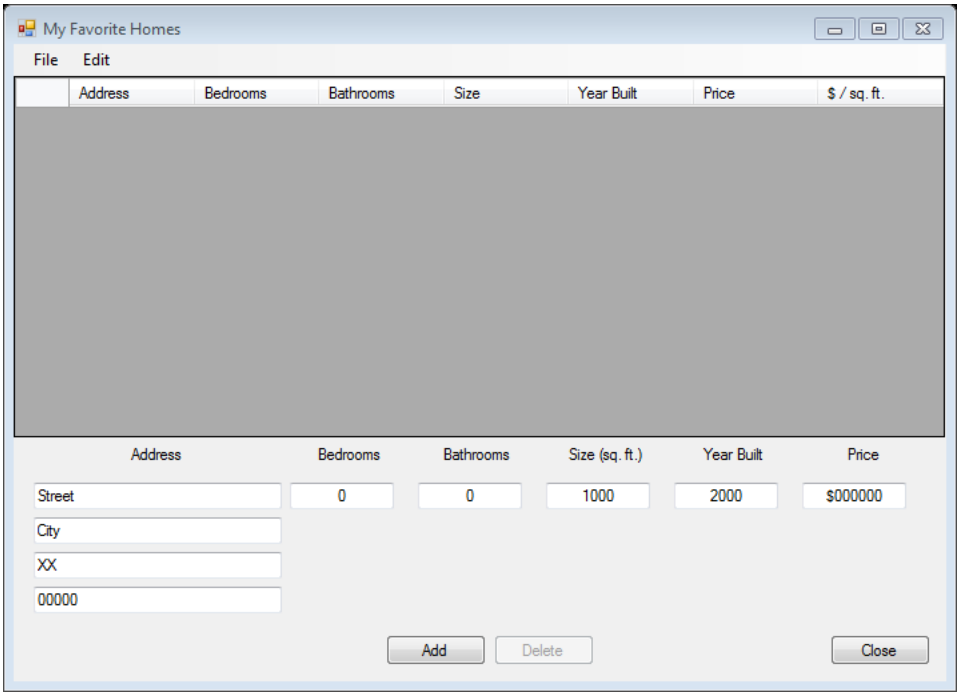

Figure 2: *Designer view of the* ***`FormFavoriteHomes`*** *form*

Program

This class is simply the initial execution point of the program. The application is started when the `FormFavoriteHomes` form is initialized and run.

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

namespace Baseline
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        private static void Main()
        {
            Application.EnableVisualStyles();
```

```
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormFavoriteHomes());
        }
    }
}
```

1.3 Output

The figures below depict the main features of the application such as adding (Figure 3), deleting (Figure 4), saving (Figure 5), and exporting (Figure 6). One feature that is available but not depicted here is the ability to clear the list from the Edit menu's 'Clear All…' option.

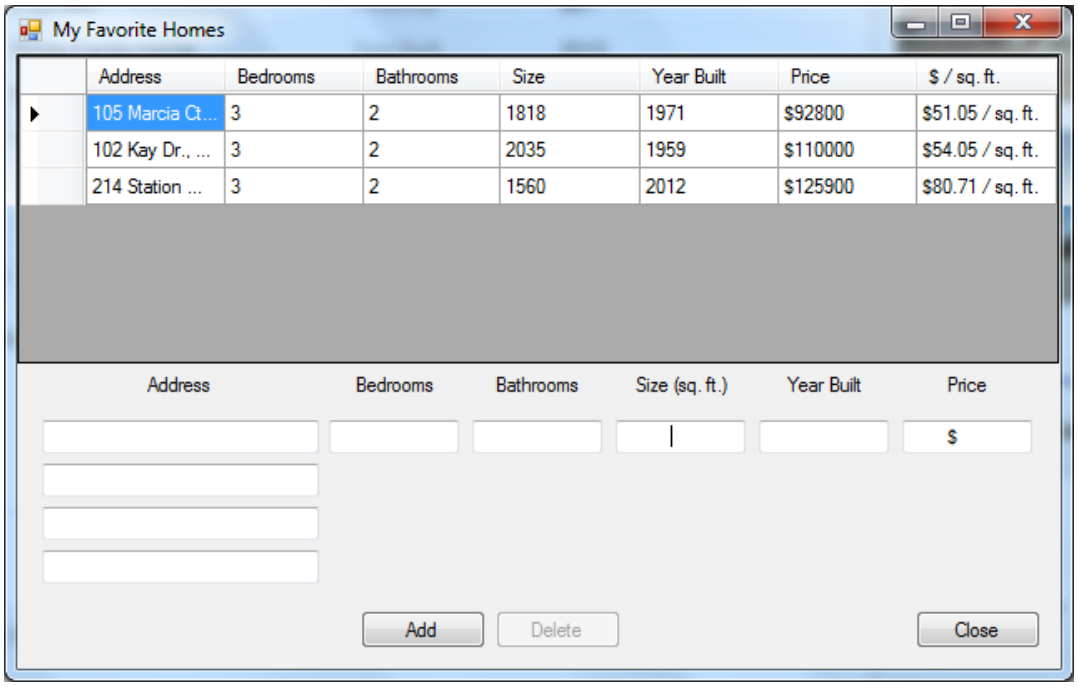

Figure 3: *List of homes added to the list*

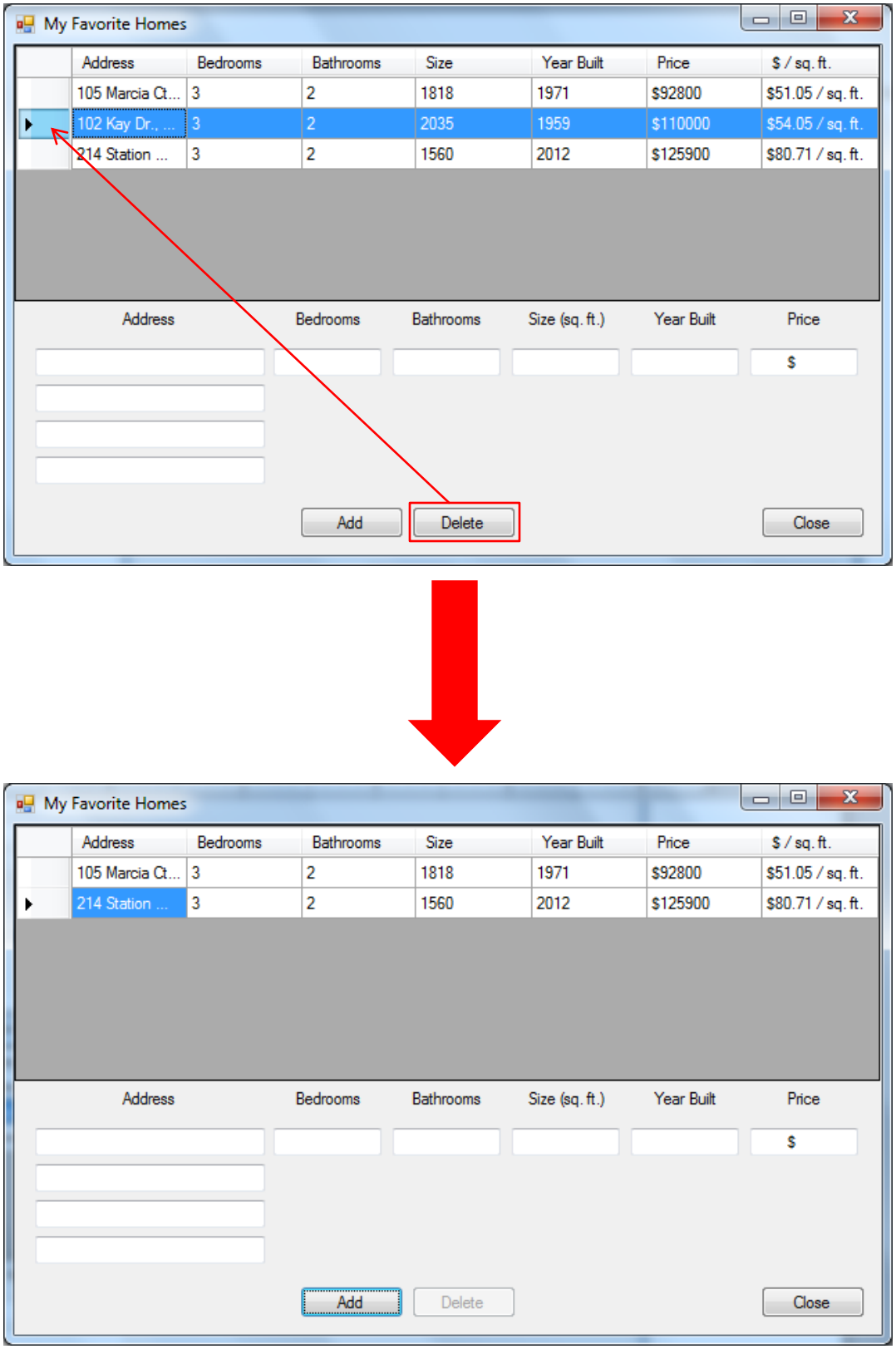

Figure 4: *Before/After deleting an item from the list*

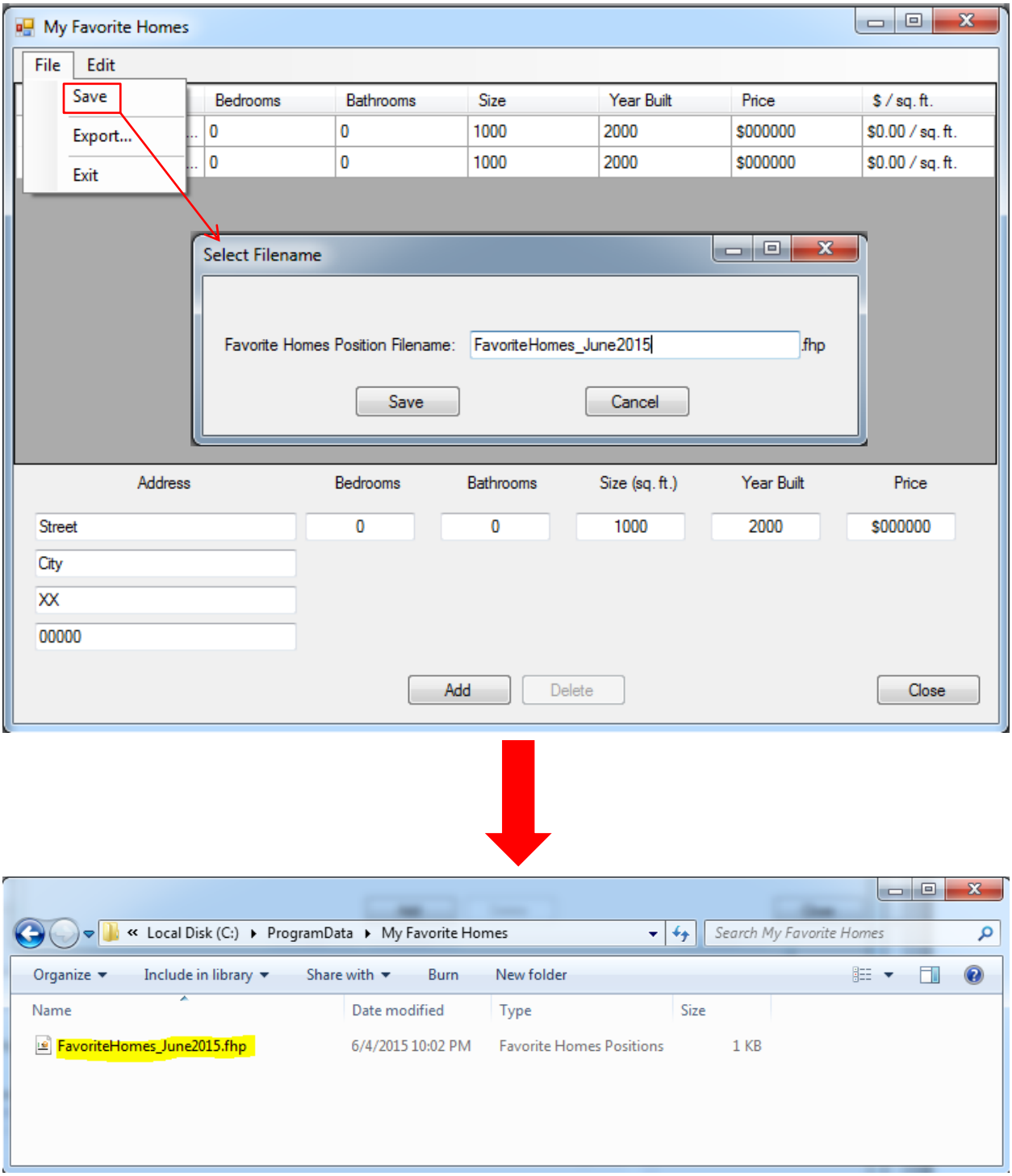

Figure 5: *'Favorite Homes Positions (*.fhp)' file created from Save.*

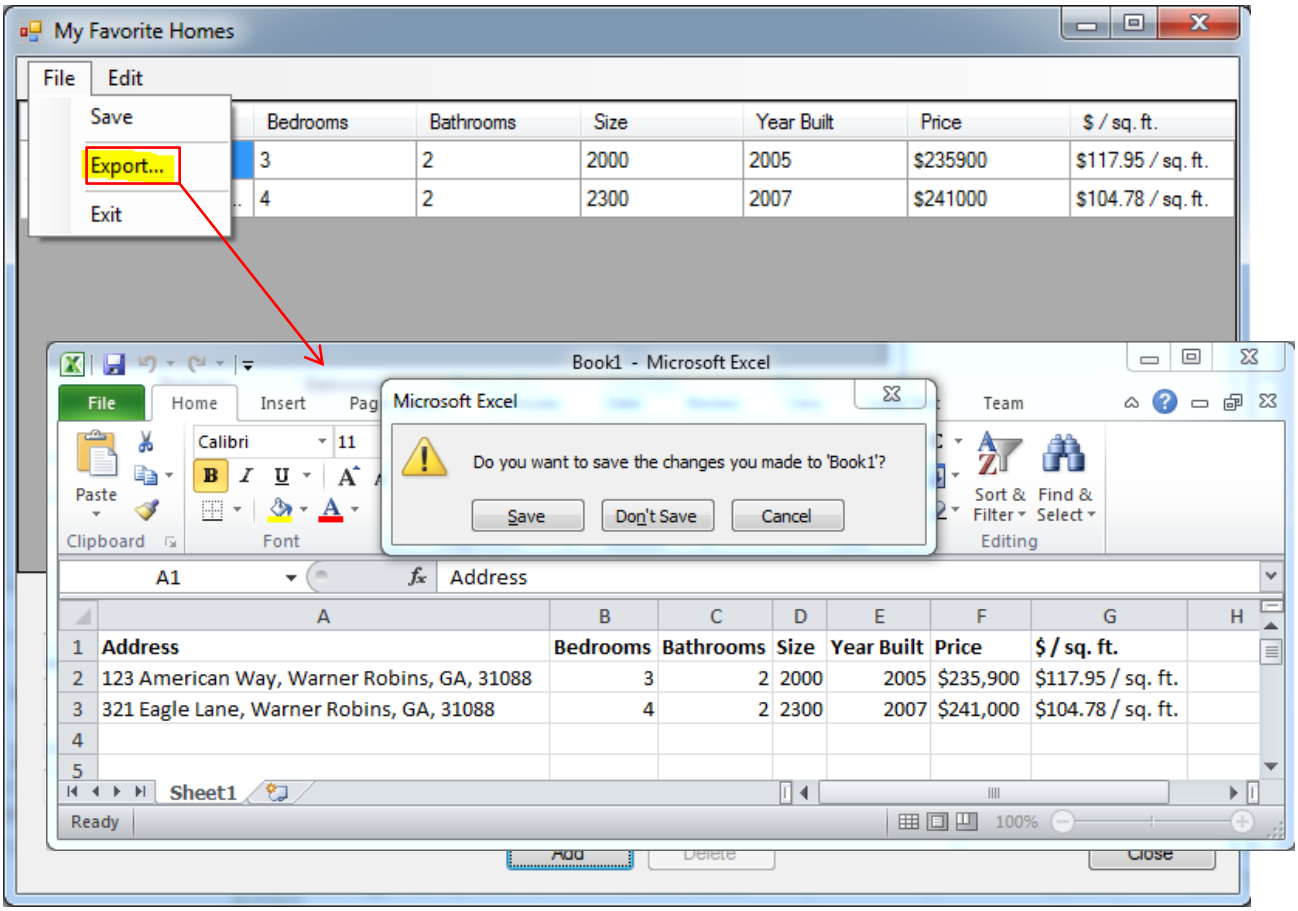

Figure 6: *Exporting the list to an Excel spreadsheet.*

2. Microkernel Pattern

The Microkernel design pattern is ideal for systems that must be adaptable for the purposes of supporting different platforms and their unique system requirements. With Microkernel, different platforms, or external servers, can provide different implementations of similar features that allow clients to use those features without requiring code changes. The microkernel supports this by providing the communication protocols used by clients to communicate with the different platforms. From here, the platforms receive the request for services and process the requests typically through some collaboration with the microkernel (i.e. file system access, customized API requests, etc.). Figure 7 presents the class diagram of the refactored solution for the '*My Favorite Homes*' app that will now utilize the Microkernel design pattern.

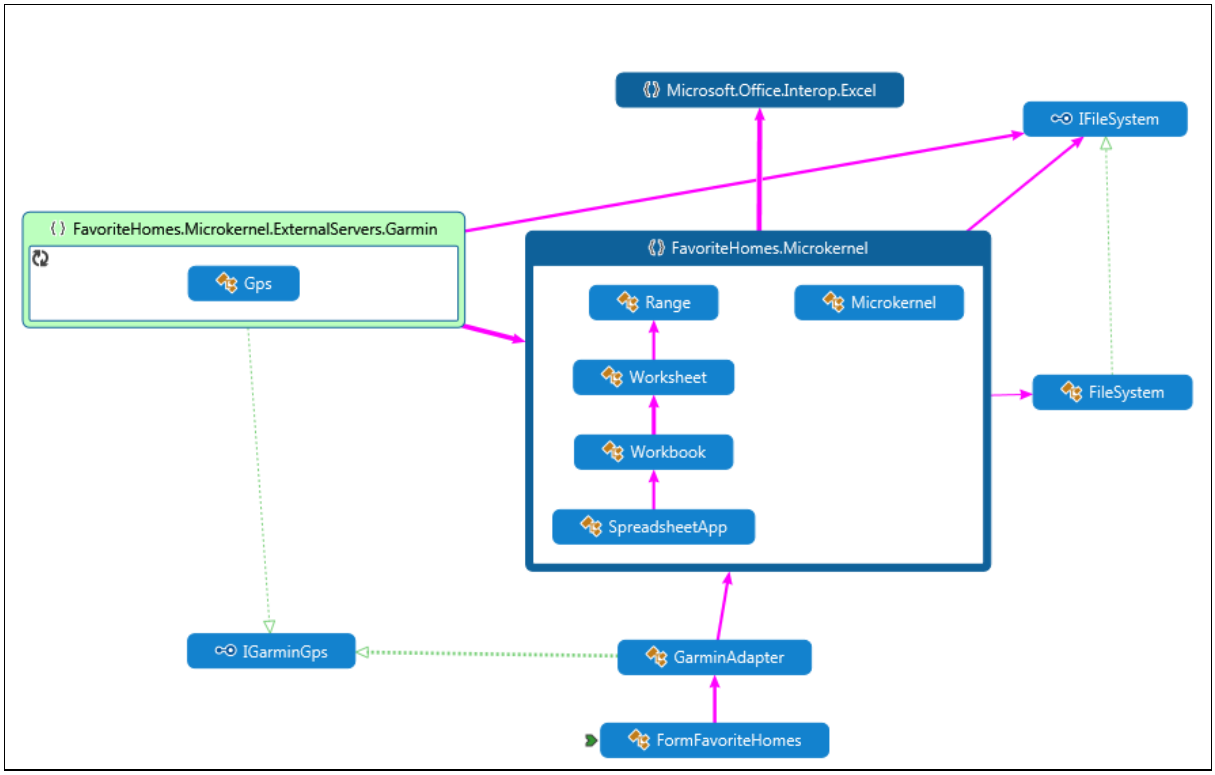

Figure 7: *Class Diagram of the 'My Favorite Homes' microkernel architecture*

The design goals include the following:

- The system should support different platforms (GPS devices).
- The system should be easily extensible and adaptable for integration of new technologies.
- The memory footprint of the application core and its provided services should be small and concise.

2.1 Architecture

Considering the design goals stated in the previous section, the Microkernel pattern will achieve these goals by decoupling a client application's dependencies on file system specifics and highly specialized API libraries (for the spreadsheet services, for example). The class diagram above

illustrates how this will be accomplished. First, the client (`FormFavoriteHomes`) will seek a service response from the GPS device (`Garmin.Gps`, `TomTom.Gps`, etc.) by initiating the call through the adapter (`GarminAdapter`). From here, the adapter will request – from the microkernel – that communications be established between the adapter and the specified GPS device. Once communications have been established, the microkernel returns the requested GPS device for use by the adapter. The adapter then sends a service request to the GPS device at which point the GPS device processes the request – typically in conjunction with service calls made through the microkernel – and returns the results to the client.

It should be noted that the client is still coupled to concrete classes in this solution, but that is beyond the scope of this project. In a typical scenario, other design patterns and principles such as Factory pattern and Dependency Injection could be utilized to remove those concrete dependencies. But this solution will still achieve the intended results with the given classes. For the sake of this project, one implementation of a simple Garmin GPS device will be presented, but the architecture easily supports other GPS devices from other manufacturers.

NOTE: For classes that are not contained within the same assembly or namespace, the fully qualified class name is used.

2.1.1 Client

The objects in this section are the client applications that depend upon services provided by specific GPS devices.

FormFavoriteHomes

As with the original prototype, this class is the main interaction point between the user and application. The main difference now is that there are no dependencies on file system or specialized library services. As seen in the class diagram, all requests for services are handled through the adapter now.

```
using System;
using System.Linq;
using System.Windows.Forms;
using FavoriteHomes.Adapters;

namespace Baseline
{
    public partial class FormFavoriteHomes : Form
    {
        /// <summary>
        /// The adapter used to handle communications with the Garmin GPS device.
        /// </summary>
        private GarminAdapter Garmin { get; set; }

        public FormFavoriteHomes()
        {
            InitializeComponent();
            Garmin = new GarminAdapter();
        }
```

```
        /// <summary>
        /// Closes the application when the Close button is pressed.
        /// </summary>
        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }

        /// <summary>
        /// Adds new homes to the list when the Add button is pressed.
        /// </summary>
        private void buttonAdd_Click(object sender, EventArgs e)
        {
            var price = Double.Parse(textBoxPrice.Text.Remove(0, 1));
            var size = Int32.Parse(textBoxSize.Text);
            var pricePerSqFt = ((price == 0.0) || (size == 0)) ? 0.0 : price/size;

            favoriteHomesDataset.dt.Rows.Add(
                textBoxStreet.Text + ", " +
                textBoxCity.Text + ", " + maskedBoxState.Text + ", " + maskedBoxZip.Text,
                Int32.Parse(textBoxBedrooms.Text),
                Int32.Parse(textBoxBathrooms.Text),
                size,
                Int32.Parse(textBoxYearBuilt.Text),
                textBoxPrice.Text,
                pricePerSqFt.ToString("C") + " / sq. ft.");
        }

        /// <summary>
        /// Deletes homes from the list when the Delete button is pressed.
        /// </summary>
        private void buttonDelete_Click(object sender, EventArgs e)
        {
            var selectedRows = dataGridView.SelectedRows;
            foreach (DataGridViewRow row in selectedRows)
            {
                dataGridView.Rows.Remove(row);
            }

        }

        /// <summary>
        /// Listener for determining if the Delete button should be enabled or not.
        /// </summary>
        private void dataGridView_RowStateChanged(object sender,
                                                  DataGridViewRowStateChangedEventArgs e)
        {
            if ((dataGridView.SelectedRows.Count > 0) &&
                (!buttonDelete.Enabled))
            {
                buttonDelete.Enabled = true;
            }
            else if ((dataGridView.SelectedRows.Count == 0) &&
                     (buttonDelete.Enabled))
            {
                buttonDelete.Enabled = false;
            }
        }

        /// <summary>
        /// Initiates the process that allows the user to save the list to a *.fhp file.
```

```
        /// </summary>
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var dlg = new SaveFhpFileDialog();

            if (dlg.ShowDialog() == DialogResult.OK)
            {
                Garmin.SaveFhpFile(dlg.Filename, favoriteHomesDataset.dt);
                MessageBox.Show("File succesfully saved!");
            }
        }

        /// <summary>
        /// Initiates the process that allows the user to export the list to a spreadsheet.
        /// </summary>
        private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var favoriteHomes = favoriteHomesDataset.dt;

            if (!favoriteHomes.Any()) return;

            Garmin.ExportToSpreadSheet(favoriteHomes);
        }

        /// <summary>
        /// Closes the application when the user selects the 'File > Exit' menu option.
        /// </summary>
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        /// <summary>
        /// Initiates the process of clearing all of the items from the list by first
        /// getting confirmation from the user and then actually clearing the items from the
        /// list if the user confirmed the clear all action.
        /// </summary>
        private void clearAllToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var result =
                MessageBox.Show("Are you sure you wish to clear all items from the list?",
                                "Clear All Items?", MessageBoxButtons.YesNo);
            if (result.Equals(DialogResult.Yes))
                favoriteHomesDataset.Clear();
        }
    }
}
```

2.1.2 Adapter

The objects in this section are the adapters that handle all service requests between the client and the GPS device (and microkernel).

GarminAdapter

This class is the specific adapter used for Garmin GPS devices. It depends upon the microkernel providing it with the correct GPS object for the sole purpose of providing an interface that allows clients to access GPS services. One thing to note is that it shares the same interface as that

implemented by the actual GPS device. This allows client applications to treat the adapter as if it were an actual instance of a GPS device.

```
using System.Data;
using FavoriteHomes.Garmin.Interfaces;

namespace FavoriteHomes.Adapters
{
    using Microkernel = Microkernel.Microkernel;

    public class GarminAdapter : IGarminGps
    {
        /// <summary>
        /// The Garmin GPS device returned by the microkernel.
        /// </summary>
        private IGarminGps Gps { get; set; }

        public GarminAdapter()
        {
            // Establish communications with the Garmin device.
            Gps = Microkernel.Instance.GetGpsDevice("Garmin");
        }

        /// <summary>
        /// Sends a service request to the GPS device for it to initiate the saving process.
        /// </summary>
        /// <param name="filename">The name of the file to be saved.</param>
        /// <param name="data">The list of homes.</param>
        /// <returns>True if the save was successful, otherwise false.</returns>
        public bool SaveFhpFile(string filename, DataTable data)
        {
            return Gps.SaveFhpFile(filename, data);
        }

        /// <summary>
        /// Sends a service request to the GPS device for it to initiate the export process.
        /// </summary>
        /// <param name="data">The list of homes.</param>
        public void ExportToSpreadSheet(DataTable data)
        {
            Gps.ExportToSpreadSheet(data);
        }
    }
}
```

2.1.3 Microkernel

The objects in this section represent the actual microkernel of the system. It provides common services that can be utilized by both external servers (GPS devices) and client adapters. It simplifies intricate calls into the file system and/or specialized libraries like the Excel interop library.

Range

This class represents a range of cells in an Excel worksheet. It is essentially an adapter class for the `Excel.Range` type from the Excel interop library. Relationally, it is the lowest level, or leaf, of the `Workbook` composite. It can represent a single cell or an entire range, or subset, of cells within the `Worksheet`. It provides access to cells by accepting two coordinates representing the two-

dimensional intersection point of a row and a column (or an entire row or column if only one parameter is specified) in a `Worksheet`.

```
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel
{
    public class Range
    {
        private readonly Excel.Worksheet _worksheet;

        private string value;

        internal Range(Excel.Worksheet worksheet)
        {
            _worksheet = worksheet;
        }

        /// <summary>
        /// Provides access to a range of cells.
        /// </summary>
        /// <param name="row">The row of the cell(s).</param>
        /// <param name="col">The column of the cell(s).</param>
        /// <returns>The range of the cell(s).</returns>
        public object this[int row = 0, int col = 0]
        {
            get { return _worksheet.Cells[row, col]; }
            set { _worksheet.Cells[row, col] = value; }
        }
    }
}
```

Worksheet

This class represents an Excel worksheet that exists with an `Excel.Workbook`. Like the `Range` class above, the `Worksheet` class is essentially an adapter class for the `Excel.Worksheet` type from the Excel interop library. Relationally, it is a composite object that lies within the scope of the `Workbook` composite and contains a `Range` of cells. There is a one-to-many relationship between the `Workbook` and the `Worksheet`.

```
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel
{
    public class Worksheet
    {
        public Range Cells { get; private set; }

        internal Worksheet(Excel.Worksheet worksheet)
        {
            Cells = new Range(worksheet);
        }
    }
}
```

Workbook

This class represents an Excel workbook that exists with an instance of the `Excel.Application`. Like the `Range` and `Worksheet` classes above, the `Workbook` class is essentially an adapter class for

the `Excel.Workbook` type from the Excel interop library. Relationally, it is the top-most composite object that encompasses one or more `Worksheet` objects. However, while true that it is the top-most composite, it still relies on an instance of the `Excel.Application` to initialize and create it.

```
using System.Collections.Generic;
using System.Linq;
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel
{
    public class Workbook
    {
        private readonly Excel.Workbook _workbook;

        public IEnumerable<Worksheet> Worksheets { get; private set; }

        /// <summary>
        /// Provides access to the active worksheet of the workbook.
        /// </summary>
        public Worksheet ActiveSheet
        {
            get { return new Worksheet(_workbook.ActiveSheet); }
        }

        /// <summary>
        /// Creates a workbook and its collection of worksheets.
        /// </summary>
        /// <param name="workbook">The workbook source.</param>
        internal Workbook(Excel.Workbook workbook)
        {
            _workbook = workbook;

            var list = new List<Worksheet>(workbook.Worksheets.Count);
            list.AddRange(from Excel.Worksheet worksheet in workbook.Worksheets
                          select new Worksheet(worksheet));
        }
    }
}
```

SpreadsheetApp

This class represents the spreadsheet application responsible for providing services for the purposes of generating spreadsheets. This class is provided by the Microkernel as an available public service to clients. This implementation wraps the specifics of the Excel interop library so that client applications can utilize basic services for creating Excel spreadsheets. Ideally, the Microkernel would also be decoupled from the Excel interop library, but the Wrapper-Façade pattern will be utilized in the next iteration to accomplish this for the system.

```
using Excel=Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel
{
    public class SpreadsheetApp
    {
        private readonly Excel.Application _app;

        public Workbook Workbook { get; private set; }
```

```
        private SpreadsheetApp()
        {
            _app = new Excel.Application() {Visible = true};
        }

        public SpreadsheetApp(int sheetsInNewWorkbook) : this()
        {
            _app.SheetsInNewWorkbook = sheetsInNewWorkbook;

            Workbook = new Workbook(_app.Workbooks.Add());

            var activeSheet = (Excel.Worksheet) _app.ActiveSheet;
            var headerRow = (Excel.Range) activeSheet.Rows[1];
            headerRow.Font.Bold = true;
        }

        public void Quit()
        {
            foreach (Excel.Worksheet worksheet in _app.ActiveWorkbook.Worksheets)
            {
                worksheet.Columns.AutoFit();
            }
            _app.Quit();
        }
    }
}
```

Microkernel

This class represents the basic stand-alone core application, or microkernel, of the system. This class is responsible for providing and establishing communications between adapters and their requested external servers. For the purposes of the project, a basic approach to establishing communications was implemented: the libraries of any external servers are searched for in a hard-coded system path and if they exist and are requested, that assembly is loaded at run-time and an instance of the external server is created and returned to the adapter. A more robust implementation could utilize framework features such as the Managed Extensibility Framework for loading the necessary external servers at run-time.

It should also be noted that because there is no logical reason to have multiple instances of the microkernel running at the same time, the microkernel was implemented as a singleton.

```
using System;
using System.Reflection;
using FavoriteHomes.Microkernel.Interfaces;
using FavoriteHomes.Microkernel.InternalServers;

namespace FavoriteHomes.Microkernel
{
    public class Microkernel
    {
        // Initialize the singleton instance
        private static readonly Microkernel _microkernel = new Microkernel();
        private IFileSystem _fileSystem;

        /// <summary>
        /// Singleton instance of the microkernel.
        /// </summary>
```

```
        public static Microkernel Instance
        {
            get { return _microkernel; }
        }

        private Microkernel()
        {
            // Prevent new instances of the Microkernel class from being created.
        }

        public IFileSystem FileSystem
        {
            get { return _fileSystem = _fileSystem ?? new FileSystem(); }
        }

        /// <summary>
        /// Communication standard provided by the microkernel that allows clients to
        /// request services from external servers (i.e. GPS devices).
        /// </summary>
        /// <param name="gpsId">Criteria used to identify the requested device.</param>
        /// <returns>The requested external server.</returns>
        public dynamic GetGpsDevice(string gpsId)
        {
            var gpsTypeName =
                string.Format("FavoriteHomes.Microkernel.ExternalServers.{0}.Gps", gpsId);

            var currentDirectory =
                FileSystem.GetDirectoryName(Assembly.GetEntryAssembly().Location);

            var binPath = FileSystem.CombinePaths(currentDirectory, @"..\..\..\bin\");

            var gpsAssemblyPath =
                string.Format(@"{0}FavoriteHomes.Microkernel.ExternalServers.{1}.dll",
                              binPath, gpsId);

            var result = Assembly.UnsafeLoadFrom(gpsAssemblyPath);

            var type = result.GetType(gpsTypeName);
            return Activator.CreateInstance(type);
        }
    }
}
```

2.1.4 External Servers

The objects in this section represent the external servers, or supported platforms, of the system. For the sake of this project, the external servers represent GPS devices that are connected to the system.

FavoriteHomes.Garmin.Interfaces.IGarminGps

This interface provides a common interface that both the external server and the adapter objects should implement. It is important that the interfaces match between the two objects so that client objects can interact with adapters in the same way they would if they were interacting with the actual GPS device itself.

```
using System.Data;

namespace FavoriteHomes.Garmin.Interfaces
```

```
{
    public interface IGarminGps
    {
        bool SaveFhpFile(string filename, DataTable data);

        void ExportToSpreadSheet(DataTable data);
    }
}
```

FavoriteHomes.Microkernel.ExternalServers.Garmin.Gps

This class is the logical representation of a Garmin GPS device. All of the GPS-supported services are implemented in this class. Any interactions with the file system and/or specialized library APIs are facilitated through the Microkernel.

```
using System;
using System.Data;
using System.Linq;
using System.Text;
using FavoriteHomes.Garmin.Interfaces;
using FavoriteHomes.Microkernel.Interfaces;

namespace FavoriteHomes.Microkernel.ExternalServers.Garmin
{
    public class Gps : IGarminGps
    {
        /// <summary>
        /// Returns the file system provided by the microkernel.
        /// </summary>
        private static IFileSystem FileSystem
        {
            get { return Microkernel.Instance.FileSystem; }
        }

        /// <summary>
        /// Responsible for saving the data in the expected binary format for Garmin
        /// GPS devices.
        /// </summary>
        /// <param name="filename">The filename to save the data to.</param>
        /// <param name="data">The favorite homes data.</param>
        /// <returns>Returns true if the save completed successfully,
        /// otherwise false.</returns>
        public bool SaveFhpFile(string filename, DataTable data)
        {
            var recordSize = 26 + // Street (26 characters)
                             2 + // formatting
                             26 + // City (26 characters)
                             2 + // formatting
                             2 + // State (2 characters)
                             2 + // formatting
                             5 + // Zip (5 characters)
                             1 + // Bedrooms (1 character)
                             1 + // Bathrooms (1 character)
                             4 + // Size (4 characters)
                             4 + // Year Built (4 characters)
                             7 // Price (7 characters)
                ;
            var homes = data.AsEnumerable().ToList();
            var buffer = new byte[1 + (recordSize * homes.Count)];
            buffer[0] = Convert.ToByte(homes.Count);
            var filePath = String.Format(@"{0}\{1}", FileSystem.DefaultSavePath, filename);
```

```
            var directoryPath = FileSystem.GetDirectoryName(filePath);
            if(!FileSystem.DirectoryExists(directoryPath))
                FileSystem.CreateDirectory(directoryPath);

            var index = 1;

            for (var i = 0; i < homes.Count; i++)
            {
                var address = homes[0].Field<string>("Address");
                var bedrooms = homes[0].Field<int>("Bedrooms");
                var bathrooms = homes[0].Field<int>("Bathrooms");
                var size = homes[0].Field<int>("Size");
                var yearBuilt = homes[0].Field<int>("Year Built");
                var price = homes[0].Field<string>("Price");

                Buffer.BlockCopy(Encoding.ASCII.GetBytes(address), 0,
                                 buffer, index, address.Length);
                index += 65;
                Buffer.BlockCopy(Encoding.ASCII.GetBytes(bedrooms.ToString()), 0,
                                 buffer, index, 1);

                index += 1;
                Buffer.BlockCopy(Encoding.ASCII.GetBytes(bathrooms.ToString()), 0,
                                 buffer, index, 1);

                index += 1;
                Buffer.BlockCopy(Encoding.ASCII.GetBytes(size.ToString()), 0,
                                 buffer, index, 4);

                index += 4;
                Buffer.BlockCopy(Encoding.ASCII.GetBytes(yearBuilt.ToString()),
                                 0, buffer, index, 4);

                index += 4;
                Buffer.BlockCopy(Encoding.ASCII.GetBytes(price), 0,
                                 buffer, index, price.Length);

                index += 7;
            }

            FileSystem.WriteBinaryFile(filePath, buffer);

            return true;
        }

        /// <summary>
        /// Utilizes the spreadsheet application provided by the microkernel to export
        /// the results in a Garmin-based format.
        /// </summary>
        /// <param name="data">The favorite homes data.</param>
        public void ExportToSpreadSheet(DataTable data)
        {
            var homes = data.AsEnumerable().ToList();

            var app = new SpreadsheetApp(1);
            var worksheet = app.Workbook.ActiveSheet;

            worksheet.Cells[1, 1] = "Address";
            worksheet.Cells[1, 2] = "Bedrooms";
            worksheet.Cells[1, 3] = "Bathrooms";
```

```
            worksheet.Cells[1, 4] = "Size";
            worksheet.Cells[1, 5] = "Year Built";
            worksheet.Cells[1, 6] = "Price";
            worksheet.Cells[1, 7] = @"$ / sq. ft.";

            for (int i = 0; i < homes.Count; i++)
            {
                var row = i + 2;

                worksheet.Cells[row, 1] = homes[0].Field<string>("Address");
                worksheet.Cells[row, 2] = homes[0].Field<int>("Bedrooms");
                worksheet.Cells[row, 3] = homes[0].Field<int>("Bathrooms");
                worksheet.Cells[row, 4] = homes[0].Field<int>("Size");
                worksheet.Cells[row, 5] = homes[0].Field<int>("Year Built");
                worksheet.Cells[row, 6] = homes[0].Field<string>("Price");
                worksheet.Cells[row, 7] = homes[0].Field<string>("PricePerSqFt");
            }

            app.Quit();
        }
    }
}
```

2.1.5 Internal Servers

The objects in this section represent the extended services, or internal servers, provided by the microkernel. Because these services can sometimes require intricate and processing-intensive code, they are not recommended for inclusion in the microkernel, but as internal servers. Typical examples of internal servers include file system architectures, graphics card interfaces, etc.

FavoriteHomes.Microkernel.Interfaces.IFileSystem

This interface provides a public interface for interacting with the file system. This decouples any dependencies on specific architectures and provides for greater flexibility and adaptability in the system architecture.

```
namespace FavoriteHomes.Microkernel.Interfaces
{
    public interface IFileSystem
    {
        string DefaultSavePath { get; }

        bool FileExists(string filePath);

        bool DirectoryExists(string directoryPath);

        string GetDirectoryName(string path);

        void CreateDirectory(string directoryPath);

        string CombinePaths(string path1, string path2);

        string[] GetDirectoryFiles(string directoryPath, string searchPattern);

        string GetFileName(string filePath);

        void Delete(string connectString);

        void WriteBinaryFile(string filePath, byte[] buffer);
```

```
    }
}
```

FavoriteHomes.Microkernel.InternalServers.FileSystem

This class is the concrete implementation of the file system architecture. All file system interactions should be called from this class (or its interface).

```
using System;
using System.IO;
using FavoriteHomes.Microkernel.Interfaces;

namespace FavoriteHomes.Microkernel.InternalServers
{
    /// <summary>
    /// Concrete implementation of the microkernal's file system services.
    /// </summary>
    public class FileSystem : IFileSystem
    {
        public string DefaultSavePath { get; private set; }

        public FileSystem()
        {
            DefaultSavePath =
                string.Format(@"{0}\{1}",
                              Environment.GetFolderPath(
                                  Environment.SpecialFolder.CommonApplicationData),
                              "My Favorite Homes");
        }

        /// <summary>
        /// Checks for the existence of a file.
        /// </summary>
        /// <param name="filePath">The file path of the file.</param>
        /// <returns>True if the file exists, otherwise false.</returns>
        public bool FileExists(string filePath)
        {
            return File.Exists(filePath);
        }

        /// <summary>
        /// Check to see if a directory exists
        /// </summary>
        /// <param name="directoryPath">The file path of the directory.</param>
        /// <returns></returns>
        public bool DirectoryExists(string directoryPath)
        {
            return Directory.Exists(directoryPath);
        }

        public string GetDirectoryName(string path)
        {
            return Path.GetDirectoryName(path);
        }

        public string CombinePaths(string path1, string path2)
        {
            return Path.Combine(path1, path2);
        }

        public void CreateDirectory(string directoryPath)
```

```
        {
            Directory.CreateDirectory(directoryPath);
        }

        /// <summary>
        /// Returns the names of files (including their paths) that match the specified search
        /// pattern in the specified directory.
        /// </summary>
        /// <param name="directoryPath">The directory to search.</param>
        /// <param name="searchPattern">The search string to match against the names of files
        /// in path. The parameter cannot end in two periods ("..") or contain two periods
        /// ("..") followed by System.IO.Path.DirectorySeparatorChar or
        /// System.IO.Path.AltDirectorySeparatorChar, nor can it contain any of the characters
        /// in System.IO.Path.InvalidPathChars.</param>
        public string[] GetDirectoryFiles(string directoryPath, string searchPattern)
        {
            return Directory.GetFiles(directoryPath, searchPattern);
        }

        /// <summary>
        /// Gets the file name for the given connection string.
        /// </summary>
        /// <param name="filePath">The connection string with the file name.</param>
        /// <returns>The file name for the given connect string.</returns>
        public string GetFileName(string filePath)
        {
            return Path.GetFileName(filePath);
        }

        /// <summary>
        /// Deletes the file from the file system.
        /// </summary>
        /// <param name="connectString">The file to delete.</param>
        /// <exception cref="InvalidOperationException">Thrown if an error occurs while
        /// deleting the file.</exception>
        public void Delete(string connectString)
        {
            try
            {
                File.Delete(connectString);
            }
            catch (Exception e)
            {
                throw new InvalidOperationException(
                    "A problem occurred trying to delete the file.", e);
            }
        }

        /// <summary>
        /// Writes the binary data buffer to the specified file.
        /// </summary>
        /// <param name="filePath">The file to create (or overwrite, if it exists).</param>
        /// <param name="buffer">The binary data buffer to be written to the file.</param>
        public void WriteBinaryFile(string filePath, byte[] buffer)
        {
            File.WriteAllBytes(filePath, buffer);
        }
    }
}
```

3. Wrapper-Façade Pattern

In the previous section, the Microkernel design pattern was applied to the system in order to support multiple platforms (GPS devices) and to create an architecture that is capable of adapting to changes in system requirements. While those design goals were achieved with the implementation of the Microkernel pattern, the microkernel itself still included a dependency on the Excel interop library. The problem with that is the microkernel would only work with systems that had the Excel application installed locally. Ideally, this dependency would be removed from the microkernel and would be a matter of installing the correct internal server for a specific implementation of a spreadsheet application (such as Office Excel, OpenOffice Calc, FreeOffice PlanMaker, etc.).

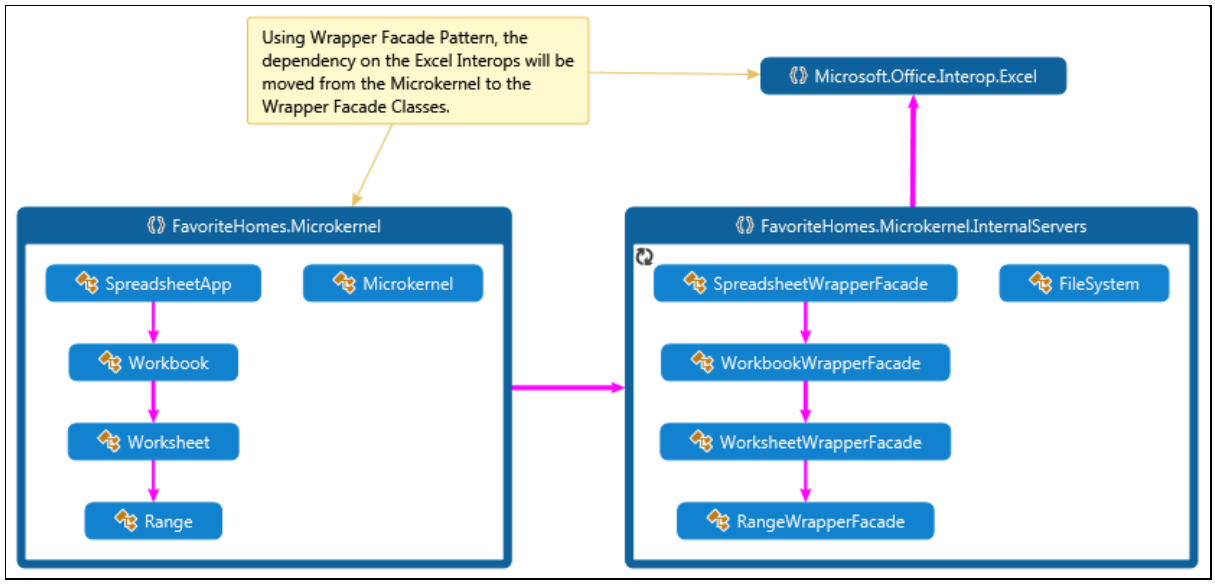

Figure 8: *Class Diagram illustrating the utilization of the Wrapper-Façade pattern for the Microkernel's spreadsheet application*

As it stands with the current design, the microkernel is rigid in that it only supports the Excel application. However, by utilizing the Wrapper-Façade pattern, the tight coupling between the microkernel and the Excel interop library can be resolved. Figure 8 presents the updated class diagram of the relationship between the microkernel and the spreadsheet application. As can be seen from the diagram, structurally, the microkernel will not change, but the implementation details will change. The spreadsheet classes (`SpreadsheetApp`, `Workbook`, `Worksheet`, and `Range`) will instead make calls to façade classes that are included as internal servers now. The façade classes will handle the specific API calls to the Excel interop library and the same can now be done for other spreadsheet applications which will allow the system to support different spreadsheet applications instead of just one. Also, because the implementation details are handled by the internal server classes, any dependencies upon specialized libraries will exist with the internal server and not the microkernel (as illustrated in the diagram above).

3.1 Architecture

In the previous design, the spreadsheet classes served as adapters for the Excel interop API so that clients could create spreadsheets. Since these adapter classes were dependent upon the Excel interop library, the system could only provide spreadsheet services as long as the Excel application was installed on the system. However, by allowing the microkernel's spreadsheet classes to access the façade classes, different spreadsheet APIs can now be easily supported.

The façade classes (`SpreadsheetWrapperFacade`, `WorkbookWrapperFacade`, `WorksheetWrapperFacade`, and `RangeWrapperFacade`) will follow a similar pattern as the original implementation of the microkernel's spreadsheet classes in that they will essentially serve as wrapper classes to encapsulate the intricate calls of the Excel interop library. A more robust design would provide an abstract parent class for other spreadsheet manufacturers to implement in order to extend the services provided by the microkernel, but the focus for this project is only on decoupling a single dependency so this will still achieve the desired design goals.

3.1.1 Wrapper-Façade (as an Internal Server)

The objects in this section represent the façade-wrapper classes for use by the microkernel's spreadsheet services. Any intricate implementation and/or low-level API details will be implemented in these classes. This allows client code to only require dependencies on the façade classes instead of specific spreadsheet APIs.

RangeWrapperFacade

This is the façade class that represents a range of cells in a worksheet. It mimics the same behavior of the previous design's `Range` class.

```
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel.InternalServers
{
    public class RangeWrapperFacade
    {
        private readonly Excel.Worksheet _worksheet;

        private string value;

        internal RangeWrapperFacade(Excel.Worksheet worksheet)
        {
            _worksheet = worksheet;
        }

        public object this[int row = 0, int col = 0]
        {
            get { return _worksheet.Cells[row, col]; }
            set { _worksheet.Cells[row, col] = value; }
        }
    }
}
```

WorksheetWrapperFacade

This is the façade class that represents a worksheet that exists within the scope of a workbook. It mimics the same behavior of the previous design's `Worksheet` class.

```
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel.InternalServers
{
    public class WorksheetWrapperFacade
    {
        private Excel.Worksheet _worksheet;
        public RangeWrapperFacade Cells { get; private set; }

        internal WorksheetWrapperFacade(Excel.Worksheet worksheet)
        {
            _worksheet = worksheet;
            Cells = new RangeWrapperFacade(worksheet);
        }

        public void AutoFitColumns()
        {
            _worksheet.Columns.AutoFit();
        }
    }
}
```

WorkbookWrapperFacade

This is the façade class that represents a spreadsheet application's workbook. It mimics the same behavior of the previous design's `Workbook` class.

```
using System.Collections.Generic;
using System.Linq;
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel.InternalServers
{
    public class WorkbookWrapperFacade
    {
        private readonly Excel.Workbook _workbook;

        public IEnumerable<WorksheetWrapperFacade> Worksheets { get; private set; }

        public WorksheetWrapperFacade ActiveSheet
        {
            get { return new WorksheetWrapperFacade(_workbook.ActiveSheet); }
        }

        internal WorkbookWrapperFacade(Excel.Workbook workbook)
        {
            _workbook = workbook;

            var list = new List<WorksheetWrapperFacade>(workbook.Worksheets.Count);
            list.AddRange(from Excel.Worksheet worksheet in workbook.Worksheets
                          select new WorksheetWrapperFacade(worksheet));
            Worksheets = list;
        }
    }
}
```

SpreadsheetWrapperFacade

This is the façade class that represents an instance of a spreadsheet application. It mimics the same behavior of the previous design's SpreadsheetApp class.

```
using Excel = Microsoft.Office.Interop.Excel;

namespace FavoriteHomes.Microkernel.InternalServers
{
    public class SpreadsheetWrapperFacade
    {
        private readonly Excel.Application _app;

        public WorkbookWrapperFacade Workbook { get; private set; }

        public SpreadsheetWrapperFacade(int sheetsInNewWorkbook)
        {
            _app = new Excel.Application
                   {
                       Visible = true,
                       SheetsInNewWorkbook = sheetsInNewWorkbook
                   };

            Workbook = new WorkbookWrapperFacade(_app.Workbooks.Add());

            var activeSheet = (Excel.Worksheet)_app.ActiveSheet;
            var headerRow = (Excel.Range)activeSheet.Rows[1];
            headerRow.Font.Bold = true;
        }

        public WorkbookWrapperFacade AddNewWorkbook()
        {
            return new WorkbookWrapperFacade(_app.Workbooks.Add());
        }

        public void Quit()
        {
            _app.Quit();
        }
    }
}
```

3.1.2 Updated Microkernel

The objects in this section simply illustrate the changes made to the pre-existing microkernel classes. Structurally and conceptually, the classes have not changed. However, the implementation details related to the spreadsheet services offered by the microkernel contain the most changes. Instead of making direct calls to the Excel interop library, the calls are now made to their equivalent façade class thereby removing any dependencies on specific spreadsheet APIs. The changes are presented below and highlighted for brevity.

Range

```
using FavoriteHomes.Microkernel.InternalServers;

namespace FavoriteHomes.Microkernel
{
    public class Range
    {
```

```
        private readonly WorksheetWrapperFacade _worksheet;

        private string value;

        internal Range(WorksheetWrapperFacade worksheet)
        {
            _worksheet = worksheet;
        }

        public object this[int row = 0, int col = 0]
        {
            get { return _worksheet.Cells[row, col]; }
            set { _worksheet.Cells[row, col] = value; }
        }
    }
}
```

Worksheet

```
using FavoriteHomes.Microkernel.InternalServers;

namespace FavoriteHomes.Microkernel
{
    public class Worksheet
    {
        public Range Cells { get; private set; }

        internal Worksheet(WorksheetWrapperFacade worksheet)
        {
            Cells = new Range(worksheet);
        }
    }
}
```

Workbook

```
using System.Collections.Generic;
using System.Linq;
using FavoriteHomes.Microkernel.InternalServers;

namespace FavoriteHomes.Microkernel
{
    public class Workbook
    {
        private readonly WorkbookWrapperFacade _workbook;

        public IEnumerable<Worksheet> Worksheets { get; private set; }

        public Worksheet ActiveSheet
        {
            get { return new Worksheet(_workbook.ActiveSheet); }
        }

        internal Workbook(WorkbookWrapperFacade workbook)
        {
            _workbook = workbook;

            var list = new List<Worksheet>(workbook.Worksheets.Count());
            list.AddRange(from WorksheetWrapperFacade worksheet in workbook.Worksheets
                          select new Worksheet(worksheet));
        }
```

```
    }
}
```

SpreadsheetApp

```
using FavoriteHomes.Microkernel.InternalServers;

namespace FavoriteHomes.Microkernel
{
    public class SpreadsheetApp
    {
        private readonly SpreadsheetWrapperFacade _app;

        public Workbook Workbook { get; private set; }

        private SpreadsheetApp()
        {
        }

        public SpreadsheetApp(int sheetsInNewWorkbook)
            : this()
        {
            _app = new SpreadsheetWrapperFacade(sheetsInNewWorkbook);
            Workbook = new Workbook(_app.Workbook);
        }

        public void Quit()
        {
            foreach (WorksheetWrapperFacade worksheet in _app.Workbook.Worksheets)
            {
                worksheet.AutoFitColumns();
            }
            _app.Quit();
        }
    }
}
```

Microkernel

There is no change from the original.

Non-Direct Activity Report

Date	Duration (minutes)	Specific Task / Activity
23-May-2015	92	Research/work on project #2.
24-May-2015	80	Research/work on project #2.
25-May-2015	154	Research/work on project #2.
26-May-2015	43	Research/work on project #2.
27-May-2015	133	Research/work on project #2.
30-May-2015	480	Research/work on project #2.
31-May-2015	420	Research/work on project #2.
2-Jun-2015	120	Research/work on project #2.
4-Jun-2015	209	Research/work on project #2.
5-Jun-2015	185	Research/work on project #2.
6-Jun-2015	75	Research/work on project #2.
Sum for Report #1	**1570**	**/ 900 (1 week @ 900/wk)**
Sum for Report #2	**1991**	**/ 900 (1 week @ 900/wk)**
Sum for Report #3		**/ 900 (1 week @ 900/wk)**
Sum for Report #4		**/ 1800 (2 weeks @ 900/wk)**
Sum for Class	**3641**	**/ 4500 (5 weeks @ 900/wk)**