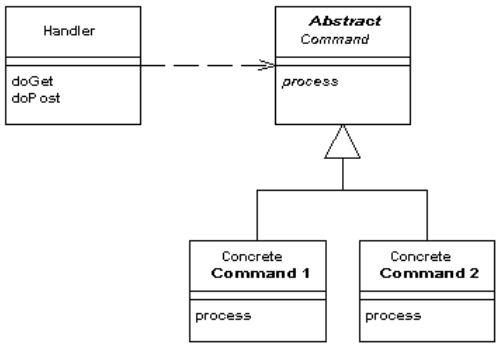
ASP.NET Web Development 2
Lab 1: Front Controller

The Front Controller pattern in a web application *"provides a centralized entry point for handling requests."* In Fowler's representation of the pattern, as shown in the figure, all requests are channelled through a single handler, or Front Controller, object. The handler then dispatches to command objects for behaviour particular to a request.



In this lab you will examine and extend a simple implementation of the Front Controller pattern in an ASP.NET Web Forms application. This example uses the same model as the *MVPDemo* example shown in the lecture, and displays either:

- a home page, if the URL is */Home.action*, or
- a page displaying a list of users, if the URL is */Users.action*

Any other URL, such as requests for an *.aspx* page, will be handled as normal.

This implementation contains the following components, most of which are located in the *Controllers* folder:

CustomHttpHandler: a custom HTTP handler which will be used by ASP.NET to handle all requests where the path matches the pattern **.action* (all other requests will be handled by the appropriate standard ASP.NET handlers). This uses a **WebRequestFactory** object to create a a **WebRequest** object, which encapsulates all the information about the request, extracted from the current *HttpContext*, and then delegates to a **FrontController** object. The custom handler is registered with the web application by adding the following to *web.config*:

```
<httpHandlers>
   <add verb="*" path="*.action"
     type="FrontControllerLab.Controller.CustomHttpHandler,
     FrontControllerLab" />
</httpHandlers>
```

FrontController: handles the request with the help of the created **WebRequest** and a **CommandRegistry** object which matches the request to the appropriate command object. The command is an instance of a class which implements the interface **IActionCommand**. FrontController then delegates to that command object.

GetUsersCommand: implements **IActionCommand** - an object of this type will be returned by the **CommandRegistry** for the URL */User.action*. This command object uses a **UserRepository** object (*Models* folder) to get the required data and a **PageNavigator** object to redirect to the appropriate view. Views are simply *.aspx* pages located within the *Views* folder. The data needs to be passed from the command into the view – this is done by storing the data in a **ViewStorage** object, which simply stores the data object (an *IEnumerable<User>* in this case, in the current HTTP context, identified by a key which is an instance of an enumeration **ViewStorageKeys**. The enum is used simply to restrict the possible keys to a known set of values, and for the moment only has one value, *ViewStorageKeys.Users*. There is also a simpler **HomePageCommand** which navigates to the appropriate view but does not need to retrieve or store data.

The View page, **UserList.aspx**, uses a **ViewStorage** object to retrieve the data which was stored by the command, and in this case binds this data to a *GridView* (it's not pretty, all we want to do here is show that the data gets to the right place!).

Exercise 1

1. Download the *FrontControllerLab* solution.

2. Test the initial version by running the web application and entering the URLs:
   http://localhost:PORT/Home.action
   http://localhost:PORT/Users.action

3. Examine the code, paying attention to the components described above.

4. Open the *file CustomHttpHandler.cs*. Right-click in the method *ProcessRequest* and select Generate Sequence Diagram (if available in your Visual Studio installation). Accept the defaults in the dialog and click OK. View the generated diagram and study the object interactions shown.

5. Repeat for the *Process* method in *GetUserCommand* and the *Page_Load* method in *UserList.aspx.cs*.

Exercise 2

The code you have been looking at may seem rather " over-engineered" simply to display some data in a web page. However, adding further functions can be done with relatively little additional code and little repetition of code. Your understanding of the code from Exercise 1 should allow you to complete the following task:

1. Add a new function to the *FrontControllerLab* application which displays details of a specific Package object which is returned by the *PackageRepository* which is provided in the Model folder. The URL for this function should provide a query parameter which specifies which

package to display. An example URL pattern would be */Package.action?packageid=3*. You will have to consider how to access the parameter in your command object. Your view page should display only the package name, description and adult price of the selected package – a set of label controls will suffice for this exercise.

2. Test the modified version by running the web application and entering the URLs:
   http://localhost:PORT/ Package.action?packageid=3
   http://localhost:PORT/ Package.action?packageid=5

3. Reflect on how much code you had to add to the application to implement the new function. Consider the advantages and disadvantages of this architecture compared to a simple Web Forms application in which code in the page code-behind files accesses the repository directly and displays the results.

4. Currently, if the URL pattern matches the path **.action*, and the action name does not match a command, then an exception is thrown. Add the functionality to display an error page in this case.

Note

This example was adapted, and simplified greatly, from the code in the book *Professional ASP.NET Design Patterns*, by Scott Millet. The example in the book is a much better model for a real implementation. It separates concerns more cleanly and makes extensive use of dependency injection. The example here is simplified to try to make the essence of the front controller pattern itself clearer.