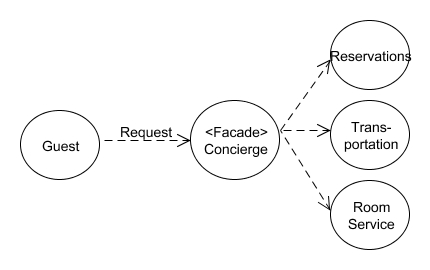
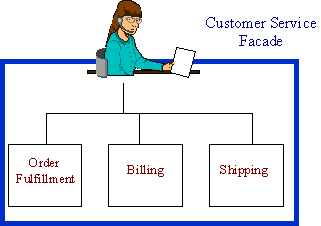
**Chapter 7 Notes – Facade Pattern**

**General Solution**

1. The Facade Pattern
2. Provides a unified interface to a set of interfaces in a subsystem.
3. Defines a higher-level interface that makes the subsystem easier to use.
4. Makes using a set of dependent classes easier.
5. A non-software example of the Facade pattern: a Concierge.



1. Another non-software example of the Facade pattern: Consumers encounter a Facade when ordering from a catalog. The consumer calls one number and speaks with a customer service representative. The customer service representative acts as a Facade, providing an interface to the order fulfillment department, the billing department, and the shipping department.



**Context/Motivation**

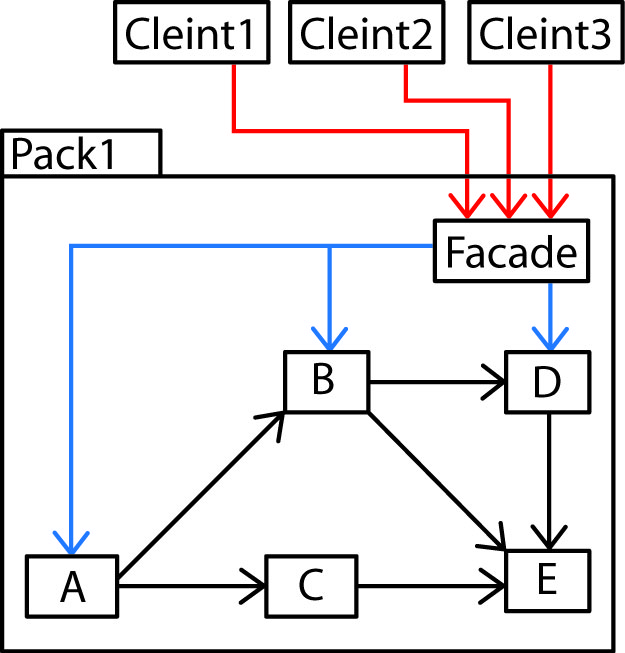
1. In the example below, there are two potential problems:
2. There is a strong dependence between the clients and the classes in the package. Thus, if the classes in the package change, then the clients must change too.
3. We may be relying on the client to use the classes in the package “in the right way.” For instance, it may be necessary for Client1 to obtain the services of an A object before using a C object.

|  |
| --- |
| Before Facade |
| E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\07-Adapter&Facade\pics\j1.jpg |

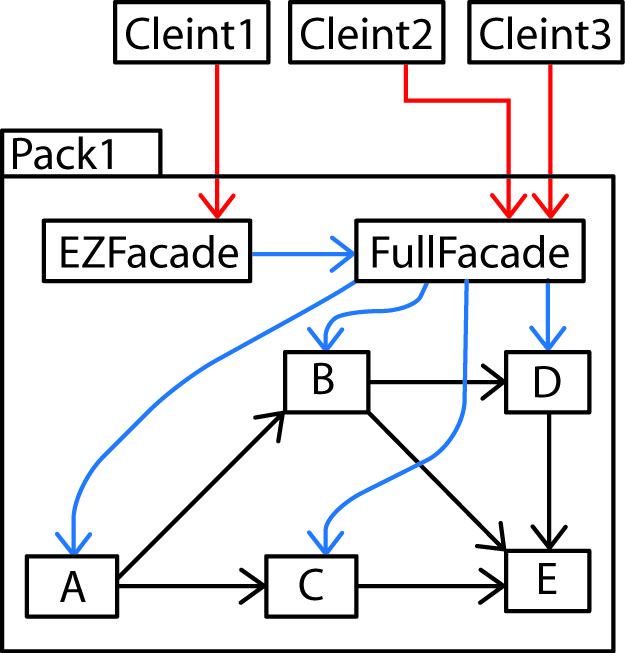
1. We can introduce a Facade which loosens the dependence as the clients now depend on the Facade instead of multiple classes. And, it is now the Facades responsibility to sequence operations in the correct way. Now, if the classes change, possibly only the Facade will need to change.

|  |
| --- |
| After Facade |
| E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\07-Adapter&Facade\pics\j2.jpg |

1. The example above also shows that some clients may still use the classes in the package directly. Clients could also be prevented from accessing the package classes by using package level scope on the classes as shown in the figure below.



1. We might also think of having an “easy” facade for some clients, perhaps with a more limited set of services as well as a “full” facade with all available services.



1. Facades can be used to handle communication between packages/layers. Although the Facades are shown in the packages, they don’t have to be.

|  |
| --- |
| Before Facades |
| E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\07-Adapter&Facade\pics\j4.jpg |
| After Facades |
| E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\07-Adapter&Facade\pics\j5.jpg |

1. Facade can be implemented with the Chain-of-Responsibility pattern. See:

<http://www.netobjectives.com/PatternRepository/index.php?title=TheFacadePattern>

1. The Facade pattern can be used to convert a legacy system to a more modern approach in an incremental fashion. See:

<http://www.netobjectives.com/PatternRepository/index.php?title=TheFacadePattern>

**Facade Pattern in Java API**

1. The Java class java.net.URL is a Facade. Behind the facade are these classes (and more): URLEncoder, URLDecoder, URLStreamHandlerFactory, URLStreamHandler, URI, URLConnection, InputStream, Socket, SocketAddress, SocketChannel, SocketFactory, SocketHandler, SocketPermission
2. The Apache web server uses RequestFacade, ResponseFacade, StandardSessionFacade.