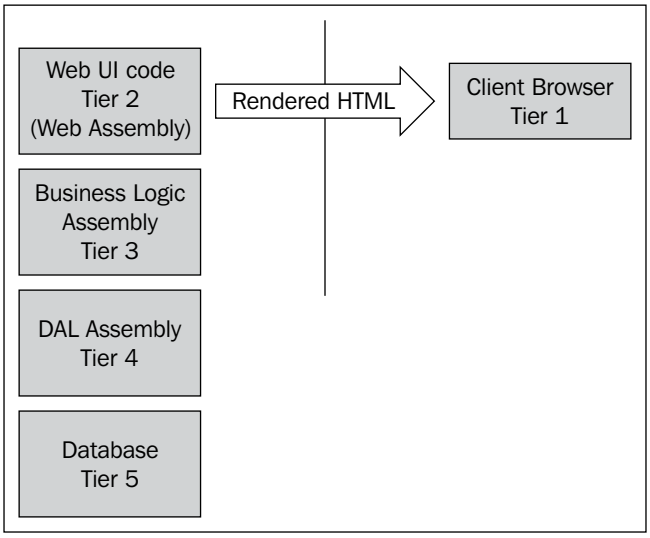
- We might want to target different physical databases instead of one single database. For example, our application might need to support MS SQL Server, Oracle and MySQL. For this to happen, we need to keep the DAL code completely out of and independent from BL so that we can easily implement a Plug and Play based architecture.
- We might want to sell individual components separately. For example, some clients might want only the DAL assembly for use in their enterprise, while some others might want both the BL and DAL assemblies, and create a custom UI. Keeping components physically separate might be financially beneficial for large commercial applications.

Note that the more we break our application into assemblies, the more performance hit we incur. However, as machine power quickly advances, a performance hit is a negligible price to pay for the overall gain in scalability as well as flexibility.

So it is beneficial to have a 5-tier architecture for relatively-large commercial applications, as this will promote better code maintenance, re-use, scalability, and adaptability.

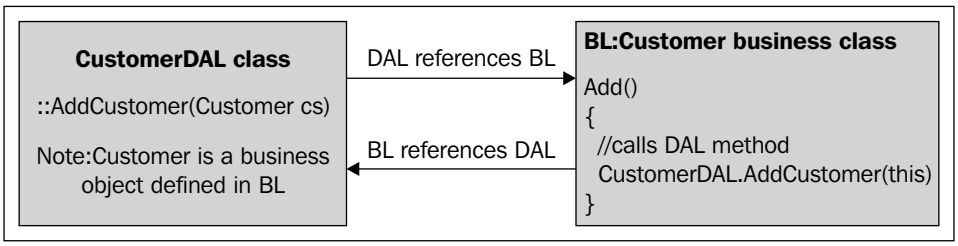
Data Transfer Objects

The following diagram shows us how the 5-tier architecture would look like in our web applications:



In this diagram, as UI, BL, and DAL are all in different tiers, we will need to add a reference to the project that is consuming another different project (say a GUI consuming a business class library project). But this brings us to a new problem—how to communicate effectively between any two tiers. Let me explain this in detail.

When the BL and DAL tiers were together in the same layer, we had no external dependencies, and we could easily call and refer to the objects of other classes by including the relevant namespaces. But now, the DAL tier will need to refer to a BL layer object in order to fill them or interact with them. But the BL tier also needs to refer the DAL tier so that BL methods can call DAL methods. This brings us to a cyclical reference issue. Refer the following diagram:



From this diagram, we can see that a need arises for both the BL and DAL to refer to each other because both are now separate projects, and also how this will lead to a cyclic reference issue, causing the compiler to immediately generate an error!

So we need to add a level of indirection to avoid this cyclical reference. There are many ways to achieve this, but one good, easy, and flexible method is to use **Data Transfer Objects**, or **DTOs**.

DTOs are simple objects with no defined methods, and having only public members. They are like carriers of data to and from the layers. Some people use strongly typed datasets for the same purpose.

The reason why we need DTOs is because passing business objects through layers is cumbersome as business objects are quite "heavy" and carry a lot of extra information with them, which is usually not needed outside the layers. So instead of passing business objects we create lightweight DTOs and make them serializable.

Let's understand how to use DTOs, through an example. We will create a 5-Tier solution for our Order Management System. Create a new VS solution, and add projects as shown here:

- `5Tier.DAL`: This class library project will encapsulate all of the data access logic. This DAL layer will **not** reference BL layer, but will communicate with it using DTOs defined in the common layer. So this DAL project references only the `5Tier.Common` project.