Generics and Custom Collections

In the above code samples, we saw the use of a lot of generics, which is a new feature starting from ASP.NET 2.0 onwards. Prior to .NET 2.0, developers used to write collection classes to hold a collection of objects. So a `Product` class would hold all of the product attributes plus the methods that perform operations on a single product such as `Update()` whereas the collection class contained methods such as `Find()`, `GetAllProducts()`, and so on.

With the introduction of generics, we can easily do away with custom collection classes. If the collection class has only standard functionality such as `Add`, `Remove()`, `GetXX()` and so on, then we can simply use generics for that. For example, the `Customer.cs` business class can have a collection object inside it say:

```
public class Customer
{
    private List<Customer> _customercollection;

}
```

So we can directly use generics instead of creating `custom` collection classes. But `custom collection` classes might be needed if we:

- Want a custom implementation of the generic `List<T>` method
- Need extra functionality, which the generic `List<T>` class doesn't offer

In these cases, we need to create our own collection class, which is basically a wrapper around the generic `List<T>` and add our own custom functions. If you look at the `Customer.cs` code or OMS code samples, you will notice that every business object (such as `Customer`) has only the following methods defined in the class:

- `Load`
- `Update`

Other methods such as `Add`, `Delete`, `FindAll`, `GetXX` and so on are defined in a `custom collection` class, because these methods operate on a "list" of entities, hence they belong to a `collection class` rather than the main entity class (like `Customer.cs`). Here is the `CustomerCollection` class for OMS:

```
public class CustomerCollection : Collection<Customer>
   {

      public CustomerCollection(): base(new List<Customer>)
      {

      }
```

```
         public bool Add(Customer c)
         {
            try
            {
               DAL.Add(c.DTO);
               c.DTO.loadStatus=LoadStatus.Ghost;
               return true;
               }
            }
         public bool Delete( Customer c)
         {
            try
            {
               DAL.Delete(c.ID);
               c.DTO.loadStatus=LoadStatus.Ghost;
               return true;
            }
         }
   public Collection<Customer> FindAll(LoadStatus loadStatus)
          {
              try
              {
                  /*
                   * Get the list of DTOs returned from the DAL and
                   * create a collection of business objects by passing
                   * in DTOs in the domain object constructor
                   */
                  Collection<CustomerDTO> dtoList =
                        CustomerDAL.GetAllCustomersNoPaging(loadStatus);
                  foreach (CustomerDTO dto in dtoList)
                  {
                      Customer customer = new Customer(dto);
                      this.Add(customer);
                  }
                  return this;
              }
              catch (Exception ex)
              {
                  //handle exception
                  throw;
              }
          }
```