**IDİSPOSABLE**

[27 Temmuz 2015](http://www.rehabayar.net/?p=264) [Mehmet Reha Bayar](http://www.rehabayar.net/?author=1) [Leave a comment](http://www.rehabayar.net/?p=264#respond)

Yazdığımız her programın en önemli noktalarından biri de performans ve bellek yönetimidir. C# programlama dilinde ram’in heap bölgesinde tutulan referance tipteki değişkenler kullanımları bittikten sonra garbage collector tarafından otomatik olarak silinirler. Ancak temizlemeye başlayacağı zaman belirsizdir.

“Nasıl olsa garbage collector bu işi yapıyor ben neden uğraşayım.” diye düşünsek de bazı durumlar da bu yapıyı bizim çalıştırmamız uygulamanın performansı açısından önem arz eder. Garbage collector u kendimiz çağırabilmemiz için öncelikle kullanacağımız classa “IDisposable” interface ini miras vermemiz ve sonrasında da implemente etmemiz gerekmektedir. Implemente işleminden sonra classımızın içine void bir method olan “Dispose” metodu gelecektir. Ram üzerinden hemen silmek istediğimiz verileri ve işlemleri bu metodun içine yazarak garbage collector un silinmesini sağlayabiliriz.

[?](http://www.rehabayar.net/?p=264)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class Product : IDisposable      {          public string ProductName { get; set; }          Product product = new Product();            public void Dispose()          {              product.ProductName = "Deneme";                product.Dispose();          }      } |

Yukarıdaki örneğimizde bir Product classı tanımlayıp bu classın IDisposable interface inden miras alıp implemente edilmesi sağladık. Sonrasında bu class a bir property tanımladık. Son olarak da bu classdan bir instance aldık. Implemente ettiğimizde gelen Dispose metodu içinde bu tanımladığımız propertye bir değer ataması gerçekleştirdik. Sonrasında da bu yarattığımız product nesnesinin tamamen ramden silinmesini sağladık. İşin mantığını anlayabilmemiz açısından ilk olarak böyle işleyişi anlatan basit bir örnek yapmak istedim.

IDisposable classmızı kullanmamızı sağlayan bir diğer yapı da using keyword ü sayesinde sağlanır. Using içinde kullandığımız herşey using in içindeki satırlar okunduktan ve işlemler gerçekleştikten sonra garbage collector çağırılarak ramden silinirler. Peki, IDisposable interface i ve using en çok nerede işimize yarayacak? Özellikle veri tabanları ile çalışırken veri tabanlarından yapılan anlık sorgular ram in heap bölgesinde oldukça fazla yer kaplarlar. Çünkü çalışma mantığı açısından programımız tüm veri tabanını ram e çeker ve sorgu yapmamızı sağlar. Ancak bu sorgu işlemi biter bitmez carbage collector hemen gelmez. Yukarıda da belirttiğimiz gibi carbage collector işlemini biz çağırmadığımız sürece otomatik yapar. Dilerseniz bu senaryoya uygun bir örnek yapalım.

[?](http://www.rehabayar.net/?p=264)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class CategoryDB      {         /// <summary>         /// Category tablosuna veri ekleme işlemi yapmaktadır.         /// </summary>         /// <param name="newCategory"></param>         /// <returns></returns>         public Category AddCategory(Category newCategory)         {             try             {                 using (var context = new Sports\_ShopEntities())                 {                     context.Category.Add(newCategory);                     context.SaveChanges();                     return newCategory;                   }             }             catch (Exception)             {                   throw;             }         } |

Yukarıdaki örneğimizde Entity Framework ile veritabanına veri ekleme işlemi gerçekleştirdik. Using içinde bulunan contex görüldüğü üzere Entity Framework üzerinden eklediğimiz veri tabanının tamamını çağırıyor. Bu da ram i çok fazla kullanacağı manasına geliyor. Çünkü veri tabanımızda çok fazla veri olabilir. Ekleme işlemi bittiği ve using in scopelarından çıktığımız anda using garbage collectoru çağırıyor ve tüm veritabanımızın ramden silinmesini sağlıyor. Bu da programımızın daha performanslı olmasını sağlıyor. Ufak bir bilgilendirme daha. Burada IDisposable interface i nerede diye sorabilirsiniz. Sports\_ShopEntities adındaki classımız bize Entity Framework sayesinde otomatik şekilde geliyor. Bu class DBContext classından DBContext de IDisposable classından miras alıyor. Yani dolaylı yoldan Sports\_ShopEntities classı da IDisposable interface ini miras almış oluyor.

Özellikle büyük projelerde performans ve bellek yönetiminin önemi büyüktür. Dolayısı ile IDisposable interface ini gerekli durumlarda kullanmak programımızın performansı açısından oldukça önemlidir.

tagged with [c#](http://www.rehabayar.net/?tag=c-2), [garbage collector](http://www.rehabayar.net/?tag=garbage-collector), [IDisposable](http://www.rehabayar.net/?tag=idisposable), [microsoft](http://www.rehabayar.net/?tag=microsoft), [microsoft türkiye](http://www.rehabayar.net/?tag=microsoft-turkiye), [object oriented programming](http://www.rehabayar.net/?tag=object-oriented-programming), [oop](http://www.rehabayar.net/?tag=oop), [using kullanımı](http://www.rehabayar.net/?tag=using-kullanimi)