



Value-Type & Reference-Type / Stack & Heap

Yüksek seviye dillerin bir iyi tarafı da, bilgisayarın arka planda nasıl çalıştığı konusunda programcının bilgi sahibi olmasına gerek kalmamasıdır. Ama bazen bilmekte de yarar var. Mesela bellekte değişkenlerin nasıl saklandığını bilmek isteyebilirsiniz. Program çalışırken değişken için bellekte değişkenin türüne göre yer ayrılır ve değişkene ihtiyacımız kalmadığında ise belleğin o bölgesi serbest bırakılır.

Bir yöntem çalışmaya başladığı zaman, içerisindeki bütün değişkenler için bellekte yer ayrılır. Bu yere stack denir. Yöntemin kullanımı sona erdiği zaman ise belleğin o bölgesi serbest bırakılır.

Şöyle bir örnek düşünelim :

```
int A, B;
```

```
long C;
```

```
string D;
```

Stack bölgesinde A ve B için 4'er byte, C için 8 byte yer ayrıldı. Peki D için ne kadar yer lazım? Bu miktar, string türündeki D değişkeninin ne kadar uzun olacağına bağlıdır. Bir string ifade çok kısa da olabilir, çok uzun da. Bu yüzden string değişkenlerin boyutu değişkendir.

Eğer string türü bir değişkene sabit bir ifade atanmışsa, boyut sabit olarak düşünülebilir. Eğer değişkenin içeriği kullanıcıdan istenen bir değer ise, veya dosyadan okunan ya da internetten alınan bir ifade ise boyutunu program çalışana kadar bilemeyiz.

Bu yüzden, string ifadeler, stack bölgesinde saklanmaz. Program çalışırken, string ifade için gereken yer heap adı verilen bölgede oluşturulur. Heap bölgesi, program çalışırken, ayırma ve serbest bırakma işlemleri yapmayı olanaklı kılan genel amaçlı bir bölgedir.

Yalnız stack ve heap bölgeleri hakkında bilinmesi gereken diğer bir gerçek var. Yukarıdaki kod bloğundaki D değişkeninin kendisi stack bölgesinde saklanır, heap bölgesinde bir alana referans içerir. D değişkeninde tutacağımız string ifadenin kendisi ise ilgili referansın gösterdiği heap bölgesindedir. Heap bölgesindeki string ifade programın akışı sırasında küçülebilir ya da büyüyebilir ama stack bölgesindeki referansı barındıran alan sabittir.

Referans derken neyi kastettiğimize biraz bakalım. Belleğin her bir hücresi numaralandırılmıştır ve bu numaralara bellek adresi denir. Program, bir bellek bölgesine erişmek isterse bu numaralardan yararlanır. Bir bellek bölgesine referans içermek demek o bölgenin adresini tutumak demektir. Yapısal programlama ile uğraşmış olanlar bunun işaretçi (pointer) demek olduğunu bilirler. Peki, içinde referans tutan bir değişken stack bölgesinde ne kadar yer kaplar? Günümüz yaygın işletim sistemleri 32-bit olduğuna göre 32 bit yani 4 byte yer kaplar.

Şimdi string ifade tutan bir değişken bellekte ne kadar yer kaplar sorusunun cevabını da bulmuş olduk. Değişkenin kendisi stack bölgesinde 4 byte yer kaplar ama string ifadenin kendisi heap bölgesinde değişken bir yere sahiptir. Kapladığı alan içerdiği ifadenin boyutuna göre değişir.

Değer-Tip & Referans-Tip (Value-Type & Reference-Type)

Yeri gelmişken referans-tip nedir, değer-tip nedir sorusuna da cevap arayalım. C#'ta yapılar değer-tiptirler. Sınıflar ise referans-tip. Bunun ne anlama geldiğini anlamak için, biraz açalım.

Aslında bir çok değer tipi daha önce kullandık. Bütün tamsayı türler .Net Framework içerisindeki yapıların aliaslarıdır. Tamsayı türdeki bir sayının gerçek değeri stack bölgesinde saklanır.

Sınıflar ise referans-tiplerdir. string veri tipi, System.String sınıfının bir aliasıdır. Yani bir sınıftır. Sınıf da referans-tiptir. Bir referans tip, heap bölgesinde bir alanı gösteren bir referans olarak stack bölgesinde saklanır.

Yapılar değer-tiptirler. Sınıflar referans-tiptirler. Bir yöntemin aşağıdaki tanımlamayı içerdiğini düşünün şimdi.

```
string Ad;
```

Yöntem çalışmaya başladığında, stack bölgesinden 4 byte'lık yer ayrılır. Bu yerde, heap bölgesinden bir alana referans barındırılabilir. Ama Ad değişkenine bir değer ataması yapılmadığı müddetçe heap bölgesinden bir alan tahsisi yapılmaz.

Şimdi bir değer atayalım.

```
Ad = "Ali";
```

Heap bölgesinde string ifade sığacak kadar yer ayrıldı ve artık stack bölgesindeki yerimiz heap bölgesinde string ifadeyi barındıran alanın adresini tutuyor, yani o bölgeye referans tutuyor.

Bir string değişkene aşağıdaki gibi hiçbir şey atamak istemezsek ne olur, inceleyelim.

```
Ad = "";
```

Şu anda hiçbir karakter içermiyor ama bir atama işlemi söz konusu olduğu için heap bölgesinden yer tahsisi yapılır.

Ad.Length ifadesinin değerini ekrana yazdırırsak 0 göreceksiniz. bu da heap bölgesinde yer içeriği boş olan bir yer ayrıldığını gösteriyor.

Çöp Toplama (Garbage Collection)

Bir string türü değişkenin değerini değiştirip heap bölgesinde bir yere referans içermemesini sağlayabiliriz. Bunu da

```
Ad = null;
```

atamasıyla yaparız. Biz bunu yaptığımız zaman ya da

```
Ad = "Veli";
```

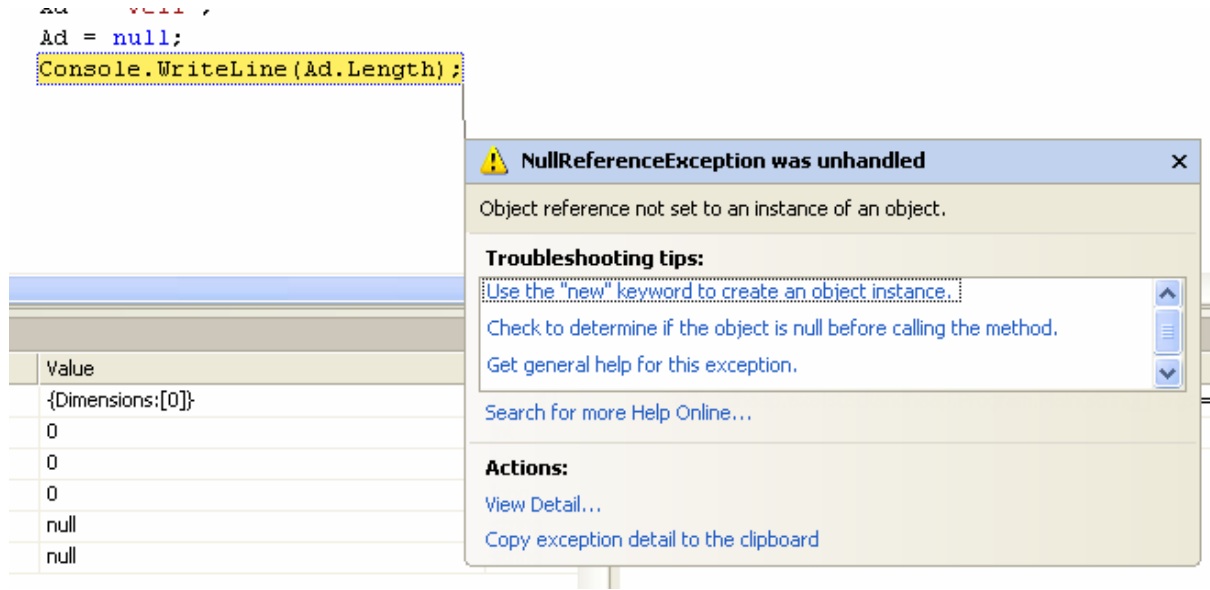
gibi başka bir değer ataması yaptığımız zaman, ilk atadığımız "Ali" ifadesi hala heap bölgesinde mevcuttur ama hiçbir referans onu göstermediği için ulaşılamaz haldedir. Bu şekilde artık kullanımı biten verileri heap bölgesinden temizleyen ve o bölgeleri kullanılabilir hale getiren çok güzel bir yapı barındırır C# ve CLR (Ortak Dil Çalışma Zamanı) : Çöp toplama (Garbage Collection). Programcı işi biten bölgeleri temizleme derdinden kurtulmuştur. Çöp Toplayıcı bunu programcının yerine yapar.

Heap bölgesinde hiçbir yere referans içermeyen yani null bir string değişkenin uzunluğunu bulmak için

`Ad.Length`

ifadesinin değerini öğrenmeye çalışırsak aşağıdaki istisna oluşur.

```
Ad = null;
Console.WriteLine(Ad.Length);
```



Value
{Dimensions:[0]}
0
0
0
null
null

Yukarıda anlatıldığı gibi, null string ile boş bir string aynı gibi görünse de aynı değildir. Bir string null'a eşitse, stack bölgesinde saklanan değer 0'a eşittir ve heap bölgesinde hiçbir yere referans içermez. Eğer bir string, boş bir string'e eşitse stack bölgesinde saklanan değer heap bölgesinde bir bölgeye referans ifade eder. Heap bölgesinde de uzunluğu 0 olan bir string için yer ayırma işlemi yapılmıştır.