T.C.
MİLLÎ EĞİTİM BAKANLIĞI

BİLİŞİM TEKNOLOJİLERİ

NESNE TABANLI PROGRAMLAMADA DEĞERLER VE BAŞVURULAR

482BK0158

Ankara, 2012

- Bu modül, mesleki ve teknik eğitim okul/kurumlarında uygulanan Çerçeve Öğretim Programlarında yer alan yeterlikleri kazandırmaya yönelik olarak öğrencilere rehberlik etmek amacıyla hazırlanmış bireysel öğrenme materyalidir.
- Millî Eğitim Bakanlığınca ücretsiz olarak verilmiştir.
- PARA İLE SATILMAZ.

İÇİNDEKİLER

AÇIKLAMALAR

KOD	**482BK0158**
ALAN	**Bilişim Teknolojileri**
DAL/MESLEK	**Veritabanı Programcılığı**
MODÜLÜN ADI	**Nesne Tabanlı Programlamada Değerler ve Başvurular**
MODÜLÜN TANIMI	Nesne tabanlı programlama dilinde değer türü ve başvuru türü değişkenleri kullanma yeterliliğinin kazandırıldığı bir öğrenme materyalidir.
SÜRE	40/24
ÖN KOŞUL	Nene Tabanlı Programlamada Sınıflar modülünü almış olmak
YETERLİK	Değer türü ve başvuru türü değişkenleri kullanabilmek
MODÜLÜN AMACI	**Genel Amaç** Bu modül ile gerekli ortam sağlandığında değer türü ve başvuru türü değişkenleri kullanabileceksiniz. **Amaçlar** **1.**Veri türü değişkenlerini kullanabileceksiniz. **2.**"ref ve "out" parametrelerini kod içerisinde kullanabileceksiniz. **3.**Kutulama (boxing) yapabileceksiniz.
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	**Ortam:** Veritabanı Programcılığı Atölyesi **Donanım:** Visual studio programını yeterli hızda çalıştırabilecek bir bilgisayar
ÖLÇME VE DEĞERLENDİRME	Modül içinde yer alan her öğrenme faaliyetinden sonra verilen ölçme araçları ile kendinizi değerlendireceksiniz. Öğretmen modül sonunda ölçme aracı (çoktan seçmeli test, doğru-yanlış testi, boşluk doldurma, eşleştirme vb.) kullanarak modül uygulamaları ile kazandığınız bilgi ve becerileri ölçerek sizi değerlendirecektir.

GİRİŞ

Sevgili Öğrenci,

Nesne yönelimli programlama dillerini kullanabilmek için sınıfları oluşturabilme ve kullanabilmenin ne kadar önemli olduğunu artık biliyorsunuz.

Sınıfları ve diğer değişken türlerini etkin ve hatasız biçimde kullanabilmek ve uygulama geliştirebilmek için oldukça önemlidir. Sınıflar ve diğer değişken türlerinin hafızada nasıl yönetildiğini anladığınızda uygulamalarınızı geliştirirken hata ayıklama sürecinde bilgi eksikliğinizden kaynaklanan zaman kayıplarından kurtulmuş olacaksınız.

Nesne Tabanlı Programlamada Sınıflar modülünde kendi sınıfınızı bildirmeyi ve bir sınıf kurucusunu new anahtar sözcüğüyle çağırarak sınıfınızın örneklerini oluşturmayı (yani nesne oluşturmayı) öğrendiniz. Ayrıca, bir kurucu kullanarak nesneye nasıl başlangıç değeri atayacağınızı gördünüz. Bu modülde ise int, double ve char gibi temel türlerin kendine has özelliklerinin sınıf türlerinden farkını öğreneceksiniz.

ÖĞRENME FAALİYETİ–1

AMAÇ

Veri türü değişkenlerini kullanabileceksiniz.

ARAŞTIRMA

- Programlama dilinde kullanılan değişken türlerini değer türü ve başvuru türü değişkenler olarak sınıflandırınız.

1. VERİ TÜRÜ DEĞİŞKENLER

Veri türü değişkenler, tanımlandığında derleyici tarafından yeterli bellek alanı ayrılan ve değişkenin saklayacağı değerin bu bellek alanında tutulduğu değişkenlerdir. Örneğin, byte türünde bir değişken tanımlanırsa derleyici bu değişkenin değerini saklamak için bize 1 bayt (8 bit) bellek alanı ayırır. Tanımlanan bu değişkene bir değer (örneğin, 135) atandığında, bu değer ayrılan 1 baytlık bellek alanına yazılır.

Başvuru türü değişkenler tanımlandığında, değer türü değişkenlerden farklı olarak derleyici tarafından hafızada değeri saklamak için bellek alanı ayrılmaz. Bunu yerine, değeri saklayacak başka bir bellek alanının adresini tutabilecek kadar küçük bir bellek alanı ayırır. Sınıflar başvuru türü değişkenlerdir. Örneğin, kendi oluşturduğumuz bir sınıf olan Bisiklet türünde bir değişken tanımlandığında Bisiklet değişkeninin değerini tutmak için değil, Bisiklet değişkeninin değerini içeren başka bir belleğin adresini tutmak için gereken bellek ayrılır. Yani değerin kendi değil, değere başvuruda bulunan bellek adresi tutulur.

1.1. Veri Türü Değişkenleri ve Sınıfları Kopyalama

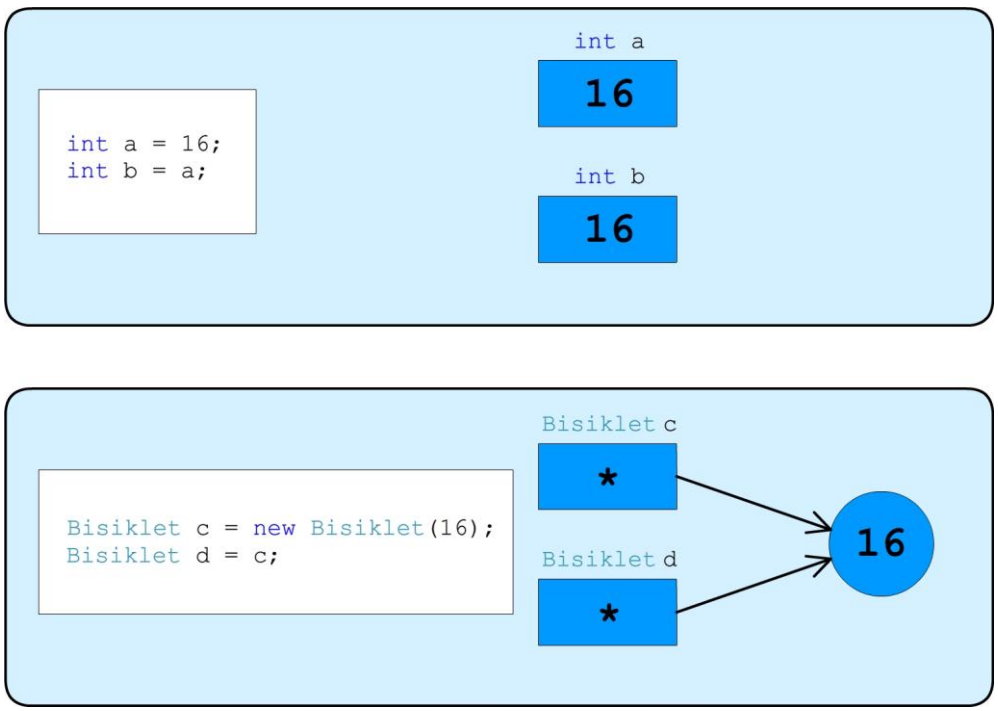
Başvuru türlerinin değer türlerinden farkını daha iyi anlatabilmek için önce iki değer türü değişkenin durumu incelenmelidir. Aşağıdaki örnekte *a* adında *int* değer türünde bir değişken tanımlanıyor ve 16 ilk değeri atanıyor. Ardından yine *int* türünde tanımlanan *b* değişkenine *a* değişkeninin değeri ilk değer olarak atanıyor. Bu iki satır kodun sonucu olarak *a* ve *b* değişkenleri aynı değeri (16) saklıyor olsalar da 16 değerini içeren iki bellek alanı bulunur. Bellek alanlarından biri *a* değişkeninin değerini, diğeri ise *b* değişkeninin değerini saklar. Üçüncü satırdaki *a* değişkeninin değerinin 1 artırılması *b* değişkenini etkilemez.

```
int a = 16;      // a değişkeni tanımlanıyor
int b = a;       // b değişkeni a'daki verinin kopyasını içerir
a++;             // a'yı artırmak b'nin değerini ekilemez
```

Programlama dilinde tanımlanan bütün sınıflar başvuru türüdür. Başvuru türü olan *c* ile *d* değişkenlerinin durumu incelenmelidir. Aşağıda *Bisiklet* türünde tanımlanmış *c* ve *d*

değişkenleri aynı bellek alanına başvurmaktadır. Yani sadece bir *Bisiklet* nesnesi vardır. *c* ve *d* değişkenlerinin her ikisi de ona başvurmaktadır. İki örnekteki farkı daha kolay anlayabilmek için aşağıdaki Şekil 1.1 oluşturulmuştur. *Bisiklet* nesnesindeki * işareti bellekteki adrese başvuruyu gösterir:

Bisiklet c = new Bisiklet(16);
Bisiklet d = c;



Şekil 1.1: Değer ve başvuru türleri arasındaki fark

1.2. Null Değerler

Şimdiye kadar tanımlanan değişkenlere genellikle aşağıdaki gibi bir ilk değer ataması yapıldı.

int s = 0;
double x = 0.0;

Başvuru değişken türü olan sınıfların yeni bir örneği oluşturulurken new anahtar sözcüğü ile nesneye bir ilk değer verildi.

Bisiklet bisiklet1 = new Bisiklet(18);

Bazı durumlarda değişkene başlangıç değeri atama sonraya bırakılabilir. Ancak bu iyi bir alışkanlık değildir. Zaten katı kuralları olan programlama dili ilk değer atanmamış bir değişkeni kullanmak istenildiğinde derleme hatası verecektir.

```
Bisiklet bisiklet1 = new Bisiklet();
Bisiklet bisiklet2;          // İlk değer atanmamış

if (bisiklet2 == null )    // Derleme hatası verir
{
        bisiklet2 = bisiklet1;
}
```

Yukarıdaki *if* ifadesi ile *bisiklet2* değişkenine bir başlangıç değeri atanıp atanmadığı kontrol edilmek istenmiştir. Fakat bu değişken eğer ilk değer atanmadı ise hangi değeri taşımaktadır? Bu sorunun cevabı null (boş anlamına gelir) adı verilen özel bir değerdir.

Programlama dilinde başvuru türü değişkenlere *null* değeri atanabilir. Bir değişkene *null* değeri atandığında, bu o değişkenin bellekte hiçbir nesneye başvurmadığı anlamına gelir.

```
Bisiklet bisiklet1 = new Bisiklet();
Bisiklet bisiklet2 = null;

if (bisiklet2 == null)
{
        bisiklet2 = bisiklet1;
}
```

1.2.1. Boş Olabilen Türler

Null değeri, başvuru türlerine başlangıç değeri vermek için kullanışlıdır. Fakat *null* değerinin kendisi bir başvurudur ve bir değer türüne atanamaz. Bu nedenle aşağıdaki ifade programlama dilinde geçerli bir ifade değildir:

```
int a = null; // geçerli değil
```

Bununla birlikte, C# bir değişkeni boş olabilen (nullable) değer türü olarak bildirmek için kullanılabilecek bir değiştirici tanımlar. Boş olabilen değer türü, orijinal değer türü ile benzer şekilde davranır fakat null değeri atanabilir. Aşağıdaki gibi değer türünün boş olabileceğini göstermek için soru işareti (?) operatörü kullanılır:

```
int? b = null; // geçerli
```

Başvuru türü test edilen yöntemle boş olabilen değişkenin null değer içerip içermediği bulunabilir:

```
if (b == null)
```

Uygun veri türü ifadesi doğrudan boş olabilen değişkene atanabilir. Aşağıdaki örnekler geçerli ifadelerdir:

```
int? b = null;
int a = 15;
b = 16;          // Bir sabiti boş olabilen türe kopyalar
b = a;           // Değer türü değişkenini boş olabilen türe kopyalar
```

Bunun tersi ise geçerli değildir. Yani boş olabilen bir değer, sıradan değer türü değişkenine atanamaz. Bu nedenle, bir önceki örnekteki a ve b değişkenlerine göre aşağıdaki ifade geçerli değildir:

```
a = b;     // Geçerli değil
```

1.2.2. Boş Olabilen Türlerin Özellikleri

Boş olabilen türlerin sahip olduğu *HasValue* (değere sahip mi?) özelliği boş olabilen bir türün bir değere mi sahip olduğunu yoksa *null* mu olduğunu kontrol edebilmeyi sağlar. Aşağıdaki gibi *value* (değer) özelliğini okuyarak bir değer içeren, boş olabilen türün değeri elde edilebilir:

```
int? b = null;
.
.
if (!b.HasValue)
  {
        b = 15;
  }
  else
  {
        Console.WriteLine(b.Value);
}
```

Yukarıdaki kod parçası, boş olabilen *b* değişkenin *HasValue* özelliği false *(! işlecine dikkat ediniz*) ise 15 değeri atar. Aksi takdirde değişkenin değerini görüntüler.

UYGULAMA FAALİYETİ

Veri türü değişkenlerini kullanınız.

İşlem Basamakları	Öneriler
➢ Nesne Tabanlı Programlama yazılımını başlatınız.	➢ Örnek projelerinizi kendinize ait bir klasör içerisinde oluşturursanız başka ortamda çalışmanız daha kolay olacaktır.
➢ *BasvuruTuruKopyalama* adında yeni bir konsol uygulaması oluşturunuz.	➢ Uygun programlama dilini seçmeyi unutmayınız.
➢ *Solution Explorer*'da *Program.cs* dosyasına çift tıklayınız.	➢ Solution Explorer penceresi görünmüyorsa *View* menüsünü kullanınız.
➢ Program sınıfında *Main* metodunu bulunuz ve bu metodun üst kısmında *Bisiklet* sınıfını aşağıdaki gibi oluşturunuz. *class Bisiklet* *{* *int hiz = 0;* *int vites = 0;* *public Bisiklet()* *{* *vites = 18;* *}* *public void Hizlan(int artis)* *{* *hiz = hiz + artis;* *}* *public void BilgileriYaz()* *{* *Console.Write("Vites:{0} ", vites);* *Console.Write("Hız: {0}", hiz);* *}* *}*	➢ Yazdığınız sınıfın Program sınıfının içinde olmasına dikkat ediniz.
➢ *Main* metodunun içerisinde *Bisiklet* nesnesi oluşturunuz ve new anahtar sözcüğü ile ilk değeri veriniz.	➢ Oluşturduğunuz nesneye anlamlı isimler veriniz (bisiklet1).
➢ İkinci bir *Bisiklet* nesnesi tanımlayın ve ilk değer olarak az önce oluşturduğunuz nesneyi atayınız.	➢ Oluşturduğunuz nesneye anlamlı isimler verin (bisiklet2). Bu sefer *new* anahtar sözcüğü ile ilk değer vermediğinizi unutmayınız.
➢ Oluşturduğunuz ilk *Bisiklet* nesnesinin *Hizlan* yöntemini çağırınız.	➢ Yönteme tamsayı türünde parametre değeri vermeyi unutmayınız.

➢ Her iki nesnenin de *BilgileriYaz* yöntemlerini çağırarak sonuçları inceleyiniz.	➢ Diğer *Bisiklet* nesnesinin *Hizlan* yöntemini çağırmadığınız hâlde bilgilerin neden aynı olduğunu tartışınız.

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet,** kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

Değerlendirme Ölçütleri		Evet	Hayır
1.	Nesne Tabanşı Programlama yazılımını doğru şekilde başlattınız mı?		
2.	Konsol uygulamasını belirli bir klasörde oluşturdunuz mu?		
3.	Program.cs dosyasını kod düzenleyicide açtınız mı?		
4.	Sınıf tanımlamasını doğru yerde yaptınız mı?		
5.	Nesne örneklerini oluşturdunuz mu?		
6.	Oluşturduğunuz nesnenin “Hizlan” yöntemini kullandınız mı?		
7.	Oluşturduğunuz nesnenin “BilgileriYaz” yöntemini kullandınız mı?		
8.	Sonuçların neden aynı olduğunu açıklayabildiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda “Hayır” şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız “Evet” ise “Ölçme ve Değerlendirme” ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. …… türü değişkeni başka bir değer türü değişkene kopyalandığında aynı değerin iki kopyası elde edilir. Ancak değişkenlerden birinin değeri değiştirildiğinde diğeri bundan etkilenmez.

2. …… türü bir değişken tanımladığınızda derleyici (belirlediğiniz türün değerini tutacak) bir bellek bloğunun adresini tutacak kadar küçük bir bellek alanı ayırır.

3. Programlama dilinde tanımlanan bütün …… başvuru türüdür.

4. Programlama dilinde tüm …… türü değişkenler boş değer alabilir.

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

5. Programlama dilinde bir değişkenin değeri boş ise bu değişkenin değeri aşağıdakilerden hangisidir?

A) Void **B)** 0 **C)** null **D)** -1

6. Aşağıdaki karakterlerden hangisi C# programlama dilinde değer türünün boş olabileceğini göstermek için kullanılır?

A) * **B)** ? **C)** // **D)** /*

7. Aşağıdaki kod parçasının kaçıncı satırı derleyici hatasına sebep olur?

```
int? x = null;      // 1. Satır
int y = 15;         // 2. Satır
x = 16;             // 3. Satır
x = y;              // 4. Satır
y = x;              // 5. Satır
```

A) 2 **B)** 3 **C)** 4 **D)** 5

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ–2

AMAÇ

Bu öğrenme faaliyeti sonunda *ref* ve *out* anahtar sözcüklerini kullanarak başvuru parametrelerini ihtiyaç duyduğunuz yerde hatasız olarak kullanabilecek, değer veya başvuru türlerini kullandığınızda belleğin *yığın* (*stack*) ve *öbek* (*heap*) bölümlerinin nasıl yönetildiğini görecek ve önemli sınıflardan birisi olan *System.Object* sınıfını kullanabileceksiniz.

ARAŞTIRMA

- Programlama dillerinde işaretçilerin (*pointer*) ne amaçla kullanıldığını araştırınız.
- Bilgisayar belleğinin hangi kısımlardan oluştuğunu ve her bir kısmın ne amaçla kullanıldığını araştırınız.

2. BAŞVURU PARAMETRELERİ

Daha önce yöntemlere parametre olarak bir değişken aktarıldığında parametrenin değişkenin kopyasının oluşturulduğu biliniyor. Bu durum parametre değer türü (*int* gibi), boş olabilen tür (*int?* gibi) ya da başvuru türü (*Bisiklet* gibi) bile böyledir. Bu da yöntem içerisinde parametrenin değerinin değiştirilmesinin aktarılan değişkenin değerini etkilemeyeceği anlamına gelir. Örneğin, aşağıdaki kodda, konsola yazılan sonuç 11 değil 10'dur. *DegerArtir* yöntemi, orijinal değişkenin (*a*) değil değişkenin kopyası olan *deger* değişkeninin değerini artırır:

```
static void DegerArtir(int deger)
{
        deger++;
}

static void Main()
{
        int a = 10;
        DegerArtir(a);
        Console.WriteLine(a); // 10 yazar
}
```

2.1. “ref” ve “out” Parametreleri

Programlama dilinde yönteme aktarılan değişkenin kendi değerini değiştirmek için ise *ref* ve *out* anahtar sözcükleri kullanılır.

2.1.1. “ref” Parametresi

Yöntem bildiriminde kullanılan bir parametrenin adının önüne *ref* yazılırsa değişkenin değeri değil, değişkenin yığındaki (*stack*’teki) adresi parametre olarak gönderilmiş olunur. Bu yapıldığında artık parametrenin değerini değiştirmek, değişkenin değerini değiştirmek anlamına gelir. Değişkeni bir *ref* parametresine aktarırken değişkenin önüne de *ref* yazılmalıdır. Bu söz dizimi, bağımsız değişkenin değişebileceğine dair görsel bir ipucu verir. Aşağıda, önceki örneğin *ref* anahtar sözcüğü kullanılarak yazılmış biçimi görülebilir:

```
static void DegerArtir(ref int deger)
{
        deger++;
}

static void Main()
{
        int a = 10;
        DegerArtir(ref a);
        Console.WriteLine(a); // 11 yazar
}
```

Bu defa, *DegerArtir* yöntemine orijinal değişkenin kopyası yerine orijinal değişkene bir başvuru tanımla. Böylece yöntemin bu başvuruyu kullanarak yaptığı her değişiklik orijinal bağımsız değişkeni de değiştirdi ve bu nedenle konsolda 11 yazdı.

2.1.2. “out” Parametresi

out anahtar sözcüğü de aslında *ref* anahtar sözcüğüne benzer. Aralarındaki fark *out* anahtar sözcüğünün değişkenlerin ilk değer verilmeden de parametre olarak aktarılmasını mümkün kılmasıdır. Örneğin ref anahtar sözcüğü kullanılan aşağıdaki kod derleme sırasında hata verir.

```
static void DegerArtir(ref int deger)
{
        deger = 10;
        deger++;
}



static void Main()
```

```
{
        int a;                          // İlk değer verilmemiş
        DegerArtir(ref a);              // Derleme hatası
        Console.WriteLine(a);
}
```

Oysa aynı örneği aşağıdaki gibi *out* anahtar sözcüğü kullanılarak denildiğinde çalışacaktır.

```
static void DegerArtir(out int deger)
{
        deger = 10;
        deger++;
}

static void Main()
{
        int a;
        DegerArtir(out a);
        Console.WriteLine(a); // 11 yazar
}
```

2.2. Yığın (Stack) ve Öbek (Heap)

Yazılan programlar ve bu programların kullandığı veriler bellek üzerinde tutulur. Değer ve başvuru türleri arasındaki farkları anlamak için verinin bellekte nasıl düzenlendiğini iyi anlamak gerekir.

İşletim sistemleri ve çalışma zamanları sık sık veriyi tutmak için kullanılan belleği her biri ayrı olarak yönetilen iki büyük parçaya ayırır. Belleğin iki büyük parçasına *yığın (stack)* ve *öbek (heap)* adı verilir. *Yığın* ve *öbek* çok farklı amaçlara hizmet eder:

- Yöntem parametreleri ve yerel değişkenleri her zaman *yığın* üzerinde oluşturulur. Yöntem bittiğinde parametreler ve yerel değişkenler için alınan bellek otomatik olarak geri verilir.

- *new* anahtar sözcüğünü kullanarak bir nesne (sınıf örneği) yaratıldığında, nesneyi oluşturmak için gerekli bellek her zaman *öbek* üzerinde oluşturulur.

- Tüm değer türleri *yığın* üzerinde yaratılır.

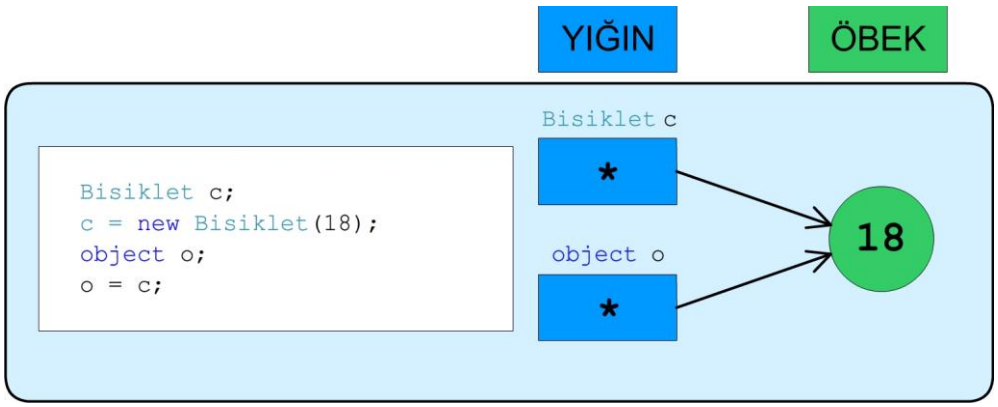
- Tüm başvuru türleri (nesneler) *öbek* üzerinde yaratılır (başvurunun kendisi de yığın üzerinde olduğu hâlde).

- Boş olabilen türler aslında başvuru türleridir ve *öbek* üzerinde yaratılır.

2.3. System.Object Sınıfı

Programlama dili *System.Object* sınıfını *object* anahtar sözcüğüyle de kullanabilmeyi sağlar. Buradan da anlaşılabileceği gibi *System.Object* önemli bir sınıftır. Başvuru türü olan herhangi bir değişkene başvuruda bulunan bir değişken oluşturmak için *System.Object* sınıfı kullanılabilir. Şekil 2.1 de bu durumdaki bellek yerleşimi gösterilmiştir.

```
Bisiklet c;
c = new Bisiklet(18);
object o;
o = c;
```



Şekil 2.1: System.Object sınıfı ile başvuru türünde bir değişkene başvuru oluşturma

UYGULAMA FAALİYETİ

"ref ve "out" parametrelerini kod içerisinde kullanınız.

İşlem Basamakları	Öneriler
➢ Nesne Tabanlı Programlama yazılımını başlatınız.	➢ Örnek projelerinizi kendinize ait bir klasör içerisinde oluşturursanız başka ortamda çalışmanız daha kolay olacaktır.
➢ *refOrnegi* adında yeni bir konsol uygulaması oluşturunuz.	➢ Programama dilini seçmeyi unutmayınız.
➢ *Solution Explorer*'de *Program.cs* dosyasına çift tıklayınız.	➢ *Solution Explorer* penceresi görünmüyorsa *View* menüsünü kullanınız.
➢ Program sınıfında *Main* yöntemini bulun ve bu yöntemin üst kısmında *ref* parametresi alan *DegerArtir* adında bir yöntem oluşturunuz.	➢ Yazdığınız sınıfın Program sınıfının içinde olmasına dikkat ediniz.
➢ *DegerArtir* yöntemi içerisinde yönteme aktarılan parametre değerini 1 artıran kodu yazınız.	
➢ Main yönteminin içerisinde tamsayı türünde bir değişken tanımlayıp bir ilk değer atayınız.	
➢ Az önce tanımladığınız değişkeni parametre olarak alan *DegerArtir* yöntemini çağırınız.	
➢ Tamsayı değişkeninin değerini ekrana yazdırıp değişkenin değerinin arttığından emin olunuz.	

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet,** kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

	Değerlendirme Ölçütleri	Evet	Hayır
1.	Nesne Tabanlı Programlama yazılımını doğru şekilde başlattınız mı?		
2.	Konsol uygulamasını belirli bir klasörde oluşturdunuz mu?		
3.	*Program.cs* dosyasını kod düzenleyicide açtınız mı?		
4.	Yöntem tanımlamasını doğru yerde yaptınız mı?		
5.	*Ref* parametre tanımını doğru yapabildiniz mi?		
6.	Yöntemi doğru şekilde çağırabildiniz mi?		
7.	Değişkenin değerinin nasıl arttığını açıklayabildiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda "Hayır" şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız "Evet" ise "Ölçme ve Değerlendirme" ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Yöntem bildirimde kullanılan ilk değeri atandığı bilinen bir parametrenin adının önüne …… yazılırsa parametre değişkenin kopyası olmak yerine onun diğer adı olur.

2. Yöntem parametreleri ve yerel değişkenleri her zaman …… üzerinde oluşturulur.

3. *new* anahtar sözcüğünü kullanarak bir nesne (sınıf örneği) yaratıldığında, nesneyi oluşturmak için gerekli bellek her zaman …… üzerinde oluşturulur.

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız.

4. ( ) Tüm değer türleri *yığın* üzerinde yaratılır.

5. ( ) Tüm başvuru türleri (nesneler) *yığın* üzerinde yaratılır (başvurunun kendisi de yığın üzerinde olduğu hâlde).

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme faaliyetine geçiniz.

ÖĞRENME FAALİYETİ–3

AMAÇ

Bu öğrenme faaliyeti sonunda programlarınız içerisinde bir değere kutulama uygulayabilecek, kutulanmış değerin gereken yerde kutulamasını kaldırabilecek ve ihtiyaç duyduğunuzda nesne dönüştürme işlemini uygun yerde hatasız olarak yapabileceksiniz.

ARAŞTIRMA

Programlama dilinde tür dönüşümü (casting) işleminin hangi durumlarda yapıldığını araştırınız.

3. BOXING VE UNBOXING

Önceki öğrenme faaliyetinde object türündeki değişkenlerin her başvuru türündeki nesneye başvurulabileceği öğrenildi. Şimdi ise object türündeki değişkenlerin bir değer türüne başvurduğunda ve object türündeki bir değişkenin bir değer türüne dönüştürüldüğünde neler olduğu incelenecektir.

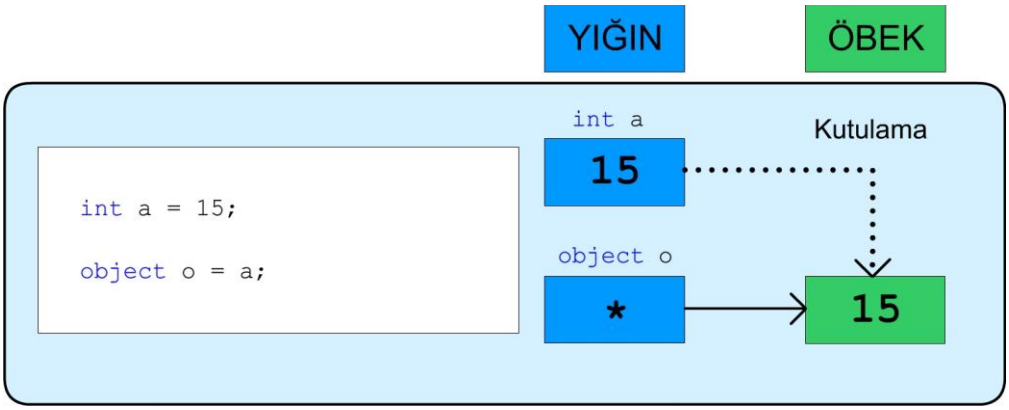
3.1. Kutulama (Boxing)

Object türündeki değişkenler bir değer türüne de başvurulabilir. Örneğin, aşağıdaki iki ifade, *a* değişkenine 15 başlangıç değerini atar ve daha sonra *o* değişkenine başlangıç değeri olarak *a* atar:

```
int a = 15;
object o = a;
```

İkinci ifadede tam olarak neler olduğunu anlamak için biraz açıklama gerekir. *a* değişkeninin değer türünde olduğunu ve yığında tutulduğunu söylenmişti. *o* değişkeni içindeki başvuru, doğrudan *a* değişkenine başvuruyorsa başvuru *yığın* üzerindeki değere olacaktır. Ancak tüm başvurular *öbek* üzerindeki nesnelere başvurmalıdır. *Yığın* üzerindeki öğelere başvuru oluşturulamaz. Bu nedenle *a* içindeki değerin bir kopyası öbek üzerinde oluşturulur ve *o* içindeki başvuru bu kopyaya olur.

Öğenin yığından öbeğe otomatik kopyalanmasına *kutulama (boxing)* adı verilir. Şekil 3.1'de bu durum gösterilmiştir.



Şekil 3.1: Kutulama (Boxing)

3.2. Kutulamayı Kaldırma (UnBoxing)

Program içerisinde bazen kutulanmış değerlerin değer türünde değişkenlere aktarılması gerekir. Bu durumda kutulamayı kaldırmak zorunlu olur. Örneğin, az önceki örnekte *int* değer türündeki *a* değişkenine, içerisinde 15 tam sayı değerine başvuruyu barındıran *object* başvuru türündeki *o* değişkeni atamaya çalışılırsa derleme hatasıyla karşılaşılır.

```
int a = 15;
object o = a;
a = o;     // Derleyici hatası
```

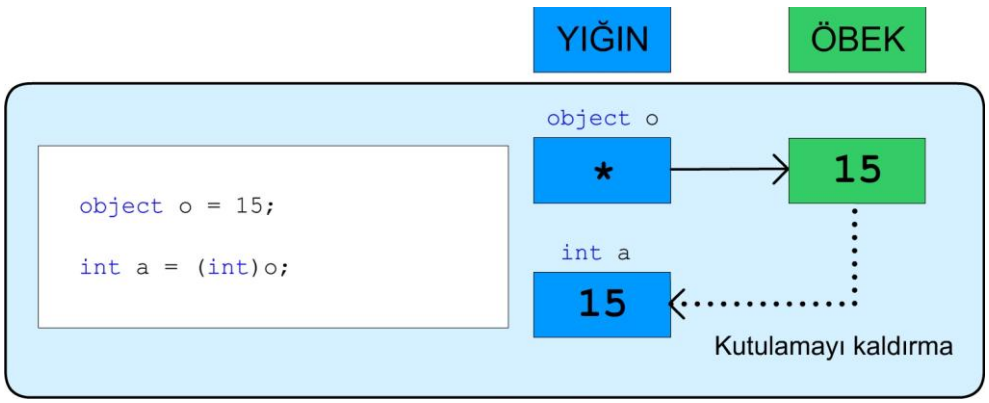
Aslında yukarıdaki kod ilk bakışta *o* değişkeni tam sayı bir değere başvuruda bulunduğu için mantıklı gibi görünebilir. Ancak object türündeki *o* değişkeninin başvuru türünde herhangi bir değişkene başvurabileceği de unutulmamalıdır. Aşağıdaki örneği incelendiğinde derleyicinin neden hata verdiği daha iyi görülebilir.

```
Bisiklet b = new Bisiklet();
int a = 10;
object o;
o = b;
a = o;   // int türüne Bisiklet başvuru türü aktarılmaya çalışılıyor
```

Peki, kutulama kaldırıldığın ne yapılmalıdır? Kutulanmış değeri elde etmek için *casting* olarak bilinen ve bir türü diğerine dönüştüren işlem kullanılır. Bu, dönüştürülmek istenen değişkenin önüne parantez içinde türün adı yazılarak yapılır.

```
int i = 15;
object o = i;          // kutulama
i = (int)o;            // kutulamayı kaldırma
```

Bu dönüştürme işlemi daha iyi anlayabilmek için aşağıdaki Şekil 3.2 oluşturulmuştur.



Şekil 3.2: Kutulamayı kaldırma (UnBoxing)

3.3. Verileri Güvenli Olarak Dönüştürme

Derleyiciler her ne kadar yapılan hatalar hemen bildirilerek işi kolaylaştırsalar da bazı durumlarda derleyiciye yol göstermek gerekir. Dönüştürme işlem kullanılarak derleyiciye bir nesne tarafından başvurulan verinin özel bir türe sahip olduğu ve bu türü kullanarak nesneye başvurmanın güvenli olduğu belirtilebilir. Programlama dili dönüştürme işlemini gerçekleştirmeye yardımcı olabilecek oldukça kullanışlı iki işleç daha sunar: *is* ve *as* işleçleri

3.3.1. "is" İşleci

Değişken türünün istenilen türde olduğunu doğrulamak için *is* işleci kullanılabilir. *is* işleci iki işlenen alır. *is* işlecinin sol tarafına doğrulamak istenilen değişkeni, sağ tarafına ise tür adı yazılır.

```
int i = 15;
object o = i;
if (o is int)
{
        i = (int)o;        // Güvenli kod: o int türündedir
}
```

Yukarıdaki örnekte de rahatlıkla görülebileceği gibi *o* değişkeninin başvurduğu değer türü eğer belirlenen türe (*int*) sahipse *is* işleci *true*, aksi takdirde *false* sonucu verir.

3.3.2. “as” İşleci

Değişken türünün hangi türde olduğunu derleyiciye anlatmak için *as* işleci kullanılabilir. *as* işleci de iki işlenen alır.

```
Bisiklet b = new Bisiklet();
object o = b;
Bisiklet c = o as Bisiklet;
```

as işleci nesneyi belirtilen türe dönüştürmeye çalışır. Başarılı ise sonuç döndürülür. Dönüştürme başarılı değilse *as* işleci *null* sonucu verir.

UYGULAMA FAALİYETİ

Kutulama (boxing) yapınız.

İşlem Basamakları	Öneriler
➢ Nesne Tabanlı programlama yazılımını başlatınız.	➢ Örnek projelerinizi kendinize ait bir klasör içerisinde oluşturursanız başka ortamda çalışmanız daha kolay olacaktır.
➢ *kutulamaOrnegi* adında yeni bir konsol uygulaması oluşturunuz.	➢ Programlama dilini seçmeyi unutmayınız.
➢ *Solution Explorer*'de *Program.cs* dosyasına çift tıklayınız.	➢ *Solution Explorer* penceresi görünmüyorsa *View* menüsünü kullanınız.
➢ *int* ve *object* türünden değişkenler oluşturarak *int* türündeki değişkene değer atayınız.	
➢ Değerlerin (*Boxing*)kutulanmasını ve (*UnBoxing*) kutulamayı kaldırma işlemlerini uygulayınız.	
➢ Değişik türden değişkenler tanımlayarak "*is*" ve "*as*" işleçlerini kullanarak değerlerin hangi türden olduğunu tespit ediniz.	

KONTROL LİSTESİ

Bu faaliyet kapsamında aşağıda listelenen davranışlardan kazandığınız beceriler için **Evet,** kazanamadıklarınız için **Hayır** kutucuklarına (X) işareti koyarak öğrendiklerinizi kontrol ediniz.

	Değerlendirme Ölçütleri	Evet	Hayır
1.	Nesne Tabanlı Programlama yazılımını doğru şekilde başlattınız mı?		
2.	Konsol uygulamasını belirli bir klasörde oluşturdunuz mu?		
3.	*Program.cs* dosyasını kod düzenleyicide açtınız mı?		
4.	*İnt* ve *object* türündedeğişken atabildiniz mi?		
5.	Değerlerin kutulamasını (*Boxing*) yapabildiniz mi?		
6.	Değerlerin kutulama kaldırmasını (*Unboxing*) yapabildiniz mi?		
7.	*is* ve *as* işleçleri ile değerlerin hangi türden olduğunu tespit edebildiniz mi?		

DEĞERLENDİRME

Değerlendirme sonunda **"Hayır"** şeklindeki cevaplarınızı bir daha gözden geçiriniz. Kendinizi yeterli görmüyorsanız öğrenme faaliyetini tekrar ediniz. Bütün cevaplarınız **"Evet"** ise "Ölçme ve Değerlendirme"ye geçiniz.

ÖLÇME VE DEĞERLENDİRME

Aşağıdaki cümlelerde boş bırakılan yerlere doğru sözcükleri yazınız.

1. Öğenin yığından öbeğe otomatik kopyalanmasına …… adı verilir.

2. Kutulanmış değeri elde etmek için …… olarak bilinen ve bir türü diğerine dönüştüren işlem kullanılır.

3. Değişken türünün istenilen türde olduğunu doğrulamak için …… işleci kullanılabilir.

4. Değişken türünün hangi türde olduğunu …… anlatmak için *as* işleci kullanılabilir.

5. Kutulanmış değeri elde etmek için ……. olarak bilinen ve bir türü diğerine dönüştüren işlem kullanılır.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru "Modül Değerlendirme" ye geçiniz.

MODÜL DEĞERLENDİRME

Aşağıdaki soruları dikkatlice okuyunuz ve doğru seçeneği işaretleyiniz.

1. Aşağıdakilerden hangisi bir başvuru türüdür?

A) double
B) int
C) int?
D) float

2. Aşağıdaki kod parçasında boşluk yerine hangisi yazılmalıdır?

```
int a = 15;
object o = a;
if (o is int)
  {
        a = (___)o;
  }
```

A) is
B) as
C) object
D) int

3. Aşağıdaki kod parçası ekrana hangisini yazar?

```
static void DegerArtir(ref int y)
{
        y++;
}

static void Main()
{
        int x = 15;
        DegerArtir(ref x);
        Console.WriteLine(x);
}
```

A) 15
B) 16
C) 14
D) 0

4. Aşağıdaki kod parçası ekrana hangisini yazar?

```
int a = 42;
object o = a;
a++;
Console.Write(o.ToString());
```

A) 0
B) 41
C) 42
D) 43

Aşağıdaki cümlelerin başında boş bırakılan parantezlere, cümlelerde verilen bilgiler doğru ise D, yanlış ise Y yazınız.

5. ( ) Değişken türünün istediğiniz türde olduğunu doğrulamak için *as* işleci kullanılır.
6. ( ) Değişkeni bir *ref* parametresine aktarırken değişkenin önüne de *ref* yazılmalıdır.
7. ( ) Bir değişkene *null* değeri atandığında, bu o değişkenin bellekte hiçbir nesneye başvurmadığı anlamına gelir.
8. ( ) *new* anahtar sözcüğü kullanılarak bir nesne (sınıf örneği) yaratıldığında, nesne oluşturmak için gerekli bellek her zaman *yığın(stack)* üzerinde oluşturulur.
9. ( ) Öğenin yığından öbeğe otomatik kopyalanmasına *kutulama (boxing)* adı verilir.
10. ( ) *int a = null;* geçerli bir ifadedir.

DEĞERLENDİRME

Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları faaliyete geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki modüle geçmek için öğretmeninize başvurunuz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ-1'İN CEVAP ANAHTARI

1	**Değer / Veri**
2	**Başvuru**
3	**sınıflar**
4	**başvuru**
5	**C**
6	**B**
7	**D**

ÖĞRENME FAALİYETİ-2'NİN CEVAP ANAHTARI

1	**ref**
2	**yığın**
3	**öbek**
4	**Doğru**
5	**Yanlış**

ÖĞRENME FAALİYETİ-3'ÜN CEVAP ANAHTARI

1	**kutulama/boxing**
2	**casting**
3	**is**
4	**derleyiciye**
5	**casting**

MÖDÜL DEĞERLENDİRMENİN CEVAP ANAHTARI

1	**C**
2	**D**
3	**B**
4	**C**
5	**Y**
6	**D**
7	**D**
8	**Y**
9	**D**
10	**Y**

KAYNAKÇA

- Sharp John, **Adım Adım Microsoft C# 2008**, Arkadaş Yayınevi, Ankara, 2009.