C, C++, C# ve Java'da enum Türleri ve Sabitleri
Benzerlikler ve Farklılıklar

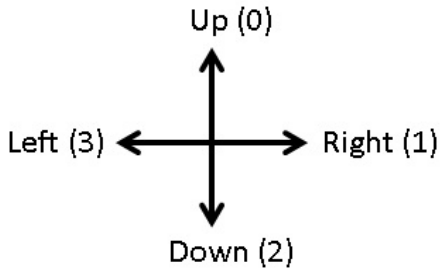
Kaan Aslan
3 Mart 2012

1. Giriş

Bazen programlarımızda haftanın günleri, yılın ayları, renkler, yönler gibi sınırlı sayıda değer alabilen çeşitli olguları ifade etmek isteriz. Örneğin top ile oynanan bir oyun programı yazacak olalım. Bu programda kullanmak üzere topu sağa, sola, yukarı ya da aşağı hareket ettiren bir fonksiyon yazmak isteyelim. Fonksiyonun parametresi topun hareket yönünü belirtiyor olsun. Sizce böyle bir fonksiyonun parametresi hangi türden olmalıdır? Şüphesiz ilk akla gelen durum parametrenin *int* gibi bir tamsayı türünden olması ve farklı yönlerin 0, 1, 2, 3 gibi bazı değerlerle temsil edilmesidir.

```
void Move(int direction)
{
     // …
}
```

Fonksiyona geçilecek geçerli argüman değerleri 0, 1, 2 ya da 3 olmalıdır. Diğer değerlerin bir anlamı yoktur.



Fonksiyonun parametre yoluyla aldığı bu değer muhtemelen fonksiyon içerisinde *switch* deyimine sokularak işlenecektir:

```
void Move(int direction)
{
     switch (direction) {
         case 0:
             //…
             break;
         case 1:
             //…
             break;
         case 2:
             //…
             break;
         case 3:
             //…
             break;
     }
     //…
}
```

Move fonksiyonunun bu biçimde bir tamsayı değer alan parametreye sahip olmasının iki sakıncası vardır. Birincisi okunabilirlikle ilgilidir. Kodu inceleyen bir kişi fonksiyonun çağrılması sırasında girilen argümanın ne anlam ifade ettiğini kolaylıkla anlayamaz. Örneğin:

```
Move(0);
//…
Move(3);
//…
Move(2);
```

Burada top hangi yönlerde hareket ettirilmektedir? Fonksiyonun bildiriminde *int* türünü gören kişi fonksiyonun herhangi bir tamsayı değerle çağrılacağını sanabilir. Oysa fonksiyon yalnızca 0, 1, 2, 3 değerlerini geçerli olarak kabul etmektedir. Topun yönünün tamsayı türünden bir parametreyle alınmasının ikinci sakıncası da hatalı çağrıma olanak verilmesidir. Şüphesiz yazılım mühendisliği bakımından, programın hatalı bir biçimde çalışmasındansa hiç derlenmemesi tercih edilen bir durumdur. Oysa örneğimizdeki *Move* fonksiyonunu programcı 100 gibi bir değerle çağırdığında derleme aşamasında bir sorun oluşmayacak fakat program da istenildiği gibi çalışmayacaktır. Koddaki böylesi bir böceğin belirlenip temizlenmesi bir hayli çaba gerektirebilmektedir.

Örneğimizdeki *Move* fonksiyonunun parametresinin *int* yerine *string* yapılması diğer bir seçeneği oluşturabilir. Bu durumda örneğin, *C#* sentaksıyla metot şöyle ifade edilebilir:

```
void Move(string direction)
{
    switch (direction) {
        case "Up":
            //…
            break;
        case "Right":
            //…
            break;
        case "Down":
            //…
            break;
        case "Left":
           //…
           break;
    }
    //…
}
```

Biz de fonksiyonu aşağıdaki gibi çağırabiliriz:

```
Move("Up"):
//…
Move("Left");
//…
Move("Down");
```

Şüphesiz bu ikinci biçim daha anlaşılabilirdir. Fakat bu yöntemin de iki önemli sakıncası vardır. Birincisi, yazısal işlemler sayısal işlemlere göre yavaştır. Örneğin iki sayının eşitliği ile iki yazının eşitliğinin sorgulanması aynı bilgisayar zamanında yapılmaz. İki yazının eşitliği sorgulanırken aslında işlem içsel olarak bir döngü içerisinde karakter karakter karşılaştırma yöntemiyle yapılmaktadır. İkincisi de, yazısal işlemler yanlış argüman kullanımına daha açıktır. Örneğin

programcı kolaylıkla *“Up”* yerine küçük harf *“up*” yazabilir. Derleme aşamasında bir denetim yapılmadığından bu durum olası bir böcek oluşmasına yol açabilir.

Şimdi sırasıyla *C*, *C++*, *C#* ve *Java*'daki enum türlerinin arasındaki benzerlikleri ve farklılıkları ele alalım.

2. C’de enum Türleri

C standartlarında (*ISO/IEC 9899:1990* ya da kısaca *C90*) *enum* türleri ve sabitleri için şu belirlemeler yapılmıştır:

1. *enum* türleri derleyicinin belirlediği bir tamsayı türüyle uyumlu olmak zorundadır. Yani *C*’de bir *enum* türü *char*, *signed char*, *unsigned char*, *int*, *unsigned int*, *long* ve *unsigned long* türlerinden biriymiş gibi ele alınmaktadır. (*ISO/IEC 9899:1990*, 6.5.5.2) *Enum* türünün uyumlu olduğu tamsayı türünün tüm *enum* sabitlerinin değerini içerebilecek bir tür olması zorunludur. (Bu koşul *C90* standartlarında unutulmuştur, *TC2*’de (*Technical Corrigendum 2*) eklenmiştir.) Tabi biz program içerisinde *enum* türünün hangi tamsayı türüyle uyumlu olduğunu anlayamayız.

2. *enum* sabitleri *int* türünden sabitlermiş gibi ele alınırlar (*ISO/IEC 9899:1990, 6.5.5.2*).

3. *enum* sabitleri bildirildikleri noktadan sonra faaliyet alanına girerler (*ISO/IEC 9899:1990, 6.1.2.1*).

4. *enum* sabitlerine = ile verilecek değerlerin tamsayı türlerinden birine ilişkin sabit ifadesi olması ve verilen değerin de *int* sınırları içerisinde kalması gerekir (*ISO/IEC 9899:1990, 6.5.5.2*).

5. *enum* türleri işlem öncesinde tamsayı türüne yükseltme kuralı gereğince *int* ya da *unsigned int* türüne dönüştürülürler (*ISO/IEC 9899:1990, 6.5.1.1*).

6. *enum* türleri kategori olarak tamsayı türlerine (*integral types*) ilişkindir. Dolayısıyla biz herhangi bir aritmetik türü *enum* türüne doğrudan atayabiliriz. Bu durumda atamanın o tamsayı türüyle enum türünün ilişkin olduğu tamsayı türü arasında yapıldığı varsayılır (*ISO/IEC 9899:1990, 6.1.2.5*).

Şimdi standartlarda belirtilen bu kuralları örneklerle açıklayalım. Aşağıdaki bildirime bakınız:

```
enum E { AA, BB, CC };

enum E x;
```

Burada *AA*, *BB* ve *CC* sabitleri *int* türündendir, dolayısıyla bunlara verilen değerlerin *int* türünün sınırları içerisinde kalan bir tamsayı türüne ilişkin sabit ifadesi olması gerekir. Fakat *enum E* türünün *int* türü olarak ele alınması zorunlu değildir. Başka bir deyişle *sizeof(enum E) == sizeof(int)* olmak zorunda değildir. *enum* türleri herhangi bir tamsayı türüyle uyumlu olacağına göre aşağıdaki atama geçersiz olabilir:

```
int a = 10;
enum E *pt;

pt = &a;           /* geçersiz olabilir, fakat olmayabilir de */
```

İki *enum* türü farklı tamsayı türlerine ilişkin olabilir. Bu durumda aşağıdaki gösterici ataması sorunludur:

```
enum E1 { AA, BB };
enum E2 { CC, DD};

enum E1 t1;
enum E2 *pt2;

pt2 = &t1;         /* geçersiz olabilir, fakat olmayabilir de */
```

C'de *enum* türleri için de *int* türüne yükseltme (*integral promotion*) kuralı uygulanmaktadır. Bir *enum* türü işleme sokulduğunda işlem öncesinde otomatik olarak *int* ya da *unsigned int* türüne yükseltilir. Böylece biz *C*'de *enum* türlerini tamsayı türleri gibi işleme sokabiliriz. Örneğin:

```
enum E1 e1;
enum E2 e2;
int result;
…
result = e1 + e2;     /* geçerli */
```

Örneğin:

```
enum E e;
long a, b;

b = e + a;
```

Burada *int* türüne yükseltme kuralı gereğince *e* önce *int* ya da *unsigned int* türüne yükseltilecek sonra *long* ile toplama işlemine sokulacaktır. *e + a* işlemin sonucu *long* türden olacaktır.

C'de bir aritmetik türü doğrudan *enum* türüne atayabiliriz. Çünkü *enum* türleri kategori olarak tamsayı türlerine ilişkindir. Böylesi bir atamada bilgi kaybının söz konusu olup olmayacağı ilgili *enum* türünün hangi tamsayı türü ile uyumlu olduğuna bağlıdır. Örneğin:

```
long a = 100000;
enum E e;

e = a;
```

Burada *int* türünün 4 byte uzunlukta olduğu sistemlerde *enum E* türü *int* ile uyumlu ise bilgi kaybı oluşmaz. Fakat örneğin *enum E* türü eğer *short* türü ile uyumlu ise bu durumda bilgi kaybı oluşacaktır.

C99 (*ISO/IEC 9899:1999*) ile *C90* (*ISO/IEC9899:1990*) standartları arasında *enum* türleriyle ilgili önemli bir farklılık yoktur.

3. C++'ta enum Türleri

C++'ın (*ISO/IEC 14882:2003*) *enum* türleri *C*'dekinden bazı bakımlardan farklıdır. Ayrıca yeni *C++* standartlarında (*ISO/IEC 14882:2011*) *enum* türlerinin sentaks ve semantik kurallarına bazı eklemeler de yapılmıştır. Biz önce eski standartlardaki klasik kurallar üzerinde duracağız. Sonra yeni *C++* standartlarında eklenen özellikleri ele alacağız.

C++ standartlarında (*ISO/IEC 14882:2003*) *enum* türleri ve sabitleriyle ilgili olarak şunlar söylenmiştir:

1. *enum* türleri kategori olarak bileşik türler (*compound types*) grubundandır ve her *enum* türü diğerlerinden farklı bir tür belirtmektedir (*ISO/IEC 14882:2003, 3.9.2*). Bu nedenle farklı *enum* türlerini parametre olarak alan aynı isimli fonksiyonlar bulunabilir ve *enum* türlerini parametre olarak alan global operatör fonksiyonları yazılabilir (*ISO/IEC 14882:2003, 5-2)*.

2. *enum* türleri işlem öncesinde otomatik olarak *int*, *unsigned int*, *long* ya da *unsigned long* türüne dönüştürülür (*ISO/IEC 14882:2003, 4.5, 5-9*).

3. *enum* türlerinden tamsayı türlerine, gerçek sayı türlerine ve *bool* türüne otomatik (*implicit*) dönüştürme vardır (*ISO/IEC 14882:2003, 4.7, 4.9, 4.12*).

4. *static_cast* operatörü ya da *C* tarzı tür dönüştürme operatörü ile bir tamsayı türü *enum* türüne ya da bir *enum* türü başka bir *enum* türüne dönüştürülebilir (*ISO/IEC 14882:2003, 5.2.9)*. Fakat gerçek sayı türleri bu operatörlerle *enum* türlerine dönüştürülemez. (*bool* türünün bir tamsayı türü olarak değerlendirildiğini anımsayınız. Bu nedenle *bool* türü de bu operatörlerle *enum* türlerine dönüştürülebilir.)

5. *enum* türlerinden adres türlerine *reinterpret_cast* operatörü ile ya da *C* tarzı tür dönüştürme operatörü ile dönüştürme yapılabilir (*ISO/IEC 14882:2003, 5.2.10)*.

6. *if* parantezinin içerisindeki ifade *enum* türünden olabilir. Bu durumda ifade otomatik olarak (*implicitly*) *bool* türüne dönüştürülür. Benzer biçimde eğer *switch* parantezi içerisindeki ifade *enum* türündense bu ifade de önce işlem öncesi otomatik olarak *int*, *unsigned int*, *long* ya da *unsigned long* türüne dönüştürülmektedir (*ISO/IEC 14882:2003, 6.4, 6.4.1, 6.4.2)*.

7. *enum* sabitleri *enum* bildiriminin dışında kullanılmışsa ilgili *enum* türündendir. Fakat *enum* bildiriminin içinde kullanılmışsa ona atanan değerin türündendir. Eğer *enum* sabitine *enum* bildirimi içerisinde değer atanmamışsa bu durumda o sabit bir önceki *enum* sabitinin türündendir. Eğer ilk *enum* sabitine programcı tarafından değer atanmamışsa ona atanmış olduğu varsayılan sıfır değerinin derleyicinin belirleyeceği herhangi bir tamsayı türünden olduğu kabul edilir (*ISO/IEC 14882:2003, 7.2-4)*.

8. Her *enum* türünün ilişkin olduğu bir tamsayı türü (*underlying integer type*) vardır. *enum* türü tamsayı türüne dönüştürüldüğünde ya da tamsayı türü *enum* türüne dönüştürüldüğünde sanki dönüştürme o tamsayı türüyle *enum* türünün ilişkin olduğu tamsayı türü arasında yapılıyormuş gibi işlemler yürütülür. Ayrıca *enum* türüne *sizeof* operatörü uygulandığında bu operatör bize *enum* türünün ilişkin olduğu tamsayı türünün uzunluğunu vermektedir. *enum* türünün ilişkin olduğu tamsayı türü herhangi bir tamsayı türü olabilir. Ancak o *enum* türünün sabitleri *int* sınırları içerisinde kalıyorsa bu tür *int* türünden büyük olamaz (*ISO/IEC 14882:2003, 7.2-4).*

Şimdi standartlarda belirtilen bu kuralların uygulamada ne anlam ifade ettiğini örneklerle açıklayalım. Öncelikle bu dilde *enum* türleri kategori olarak bir tamsayı türü olarak sınıflandırılmamıştır. *C++*'ta her *enum* türü ayrı bir tür belirtir. Dolayısıyla farklı *enum* türlerini parametre olarak alan aynı isimli fonksiyonlar bildirilebilir. Örneğin:

```
enum E1 {XX, YY};

enum E2 {ZZ, MM};

enum E3 {KK, LL};
```

```
void foo(E1 e1)
{
    //…
}

void foo(E2 e2)
{
    //…
}

void foo(E3 e3)
{
    //…
}
```

foo fonksiyonlarının imzaları (*signatures*) birbirlerinden farklıdır, dolayısıyla *overload* edilebilirler. *C++* 'ta *enum* türleri için global operatör fonksiyonları yazılabilir:

```
E3 operator +(E1 e1, E2 e2)
{
    return static_cast<E3>(e1 + e2);
}
```

enum türlerinin işlem öncesinde otomatik olarak tamsayı türlerine yükseltilmesi işlemlerde onları tamsayı gibi kullanabileceğimiz anlamına gelir. Örneğin biz bir *enum* türü ile bir tamsayı türünü ya da iki *enum* türünü aritmetik işlemlere sokabiliriz. Bu durumda bu *enum* türlerinin ilişkin oldukları tamsayı türleri dikkate alınarak dönüştürme yapılır. Örneğin *E1* ve *E2* birer *enum* belirtiyor olsun:

```
E1 e1;
E2 e2;
int result;

result = e1 + e2;
```

işlemi geçerlidir. *E1* ve *E2* türlerinin ilişkin olduğu tamsayı türleri dikkate alınarak önce *e1* ve *e2* değerleri *int*, *unsigned int*, *long* ya da *unsigned long* türüne dönüştürülür, sonra işleme sokulur. Sonuç bu türlerden birine ilişkin çıkacaktır. Sonucun *int* türüne atanması sırasında bilgi kaybı oluşabilir ya da oluşmayabilir. Benzer biçimde bir *enum* türüyle bir tamsayı türü ya da gerçek sayı türü de işleme sokulabilir.

enum türlerini biz bir tamsayı türüne, gerçek sayı türüne ya da *bool* türüne doğrudan atayabiliriz. Eğer *enum* değeri sıfır dışı bir değere sahipse dönüştürmeden *true*, sıfır değerindeyse dönüştürmeden *false* elde edilir. Örneğin *E* bir *enum* türü olmak üzere:

```
E e;
int a;
double b;
bool c;

//…

a = e;          // geçerli
b = e;          // geçerli
c = e;          // geçerli
```

Eğer tamsayı ya da *bool* türünden bir değeri *enum* türüne atamak istersek bunu ancak tür dönüştürme operatörleriyle yapabiliriz. Bunun için *C* tarzı tür dönüştürme operatörü, fonksiyonel tarzdaki tür dönüştürme operatörü ya da *static_cast* operatörü kullanılabilir. Örneğin:

```
int a = 10;
E e;

e = a;                          // geçersiz!
e = static_cast<E>(a);          // geçerli
```

Benzer biçimde bir *enum* türünden diğerine de tür dönüştürme operatörleriyle dönüştürme yapılabilir. Örneğin:

```
E1 e1;
E2 e2;
//...

e1 = e2;                        // geçersiz!
e1 = static_cast<E1>(e2);       // geçerli
```

C++ standartlarına göre gerçek sayı türlerinden *enum* türlerine tür dönüştürme operatörü kullanılsa bile dönüştürme tanımlı değildir.[1] Örneğin:

```
double a;
E e;
//...

e = static_cast<E>(a);          // geçersiz!
```

C++'ta da tıpkı *C*'de olduğu gibi bir *enum* türü derleyici tarafından içsel olarak sanki o *enum* türünün ilişkin olduğu tamsayı türündenmiş gibi ele alınmaktadır. Örneğin bir *enum* türünden nesne için o *enum* türünün ilişkin olduğu tamsayı türü kadar yer ayrılır. Tür dönüştürmeleri sırasında da bilgi kaybının oluşup oluşmaması *enum* türünün ilişkin olduğu tamsayı türü ile ilgilidir.

C++'ın yeni standartlarında (*ISO/IEC 14882:2011*) *enum* türlerine yeni eklemeler yapılmıştır. Normal olarak bir *enum* türünün sabitleri, *enum* hangi faaliyet alanındaysa o faaliyet alanında kabul edilir. Örneğin:

```
enum Color {
    Red, Green, Blue
};
```

Burada eğer *Color* isimli *enum* global isim alanına yerleştirilmişse *Red*, *Green* ve *Blue* simleri de global isim alanındadır. e*num* sabitlerinin bu özellikleri isim çakışması oluşturabilmektedir. Eğer biz *enum* sabitlerini başka bir faaliyet alanında gizlemek istiyorsak *enum* türünü bir isim alanına ya da bir sınıfa yerleştirmeliyiz. Örneğin:

```
struct Color {
    enum Color {
        Red, Green, Blue
    };
};
```

İşte *C++ 2011*'deki en önemli yenilik kendi faaliyet alanlarına sahip *enum* (*scoped enumeration*) türleridir. Yeni *enum* türleri *enum class* ya da *enum struct* anahtar sözcükleriyle bildirilir. Bu iki bildirim arasında bir farklılık yoktur. Örneğin:

```
enum class Day {
    Sunday, Monday, Tuesday, Wednesday,
    Thursday, Friday, Saturday
};

enum struct Color {
    Red, Green, Blue
};
```

Bu yeni *enum* türlerine ilişkin *enum* sabitlerinin faaliyet alanı *enum* bloğu ile sınırlıdır. Bu *enum* sabitleri blok dışından ancak :: operatörü ile niteliklendirilerek kullanılabilir. Örneğin:

```
Day day;

day = Monday;             // geçersiz!
day = Day::Monday;        // geçerli
```

Bu yeni *enum* türleri ile tamsayı ve gerçek sayı türleri arasında ve *bool* türü arasında doğrudan (*implicit*) dönüştürme yoktur. Örneğin:

```
Day day = Day::Monday;
int result;

result = day;             // geçersiz!
result = (int) day;       // geçerli
```

Yeni *enum* türleri işlem öncesinde tamsayı türlerine de dönüştürülmezler:

```
Day day = Day::Wednesday;
Color color = Color::Green;
int result;

result = day + color;     // geçersiz!
if (day > color) {        // geçersiz!
    //...
}

if (day) {                // geçersiz! bool türüne de doğrudan dönüştürme yok!
    //...
}
```

Yeni *enum* türlerinin ilişkin olduğu tamsayı türleri *: <tür>* sentaksıyla belirlenebilmektedir. Örneğin:

```
enum class Test : long {
    AA, BB, CC
};
```

enum türünün ilişkin olduğu tamsayı türü bildirimde belirtilmeyebilir. Bu durumda ilişkin olunan tür *int* kabul edilmektedir. Yeni enum türlerinin *sabitleri* ilişkin olunan tamsayı türlerinin sınırlarını aşamaz. Örneğin:

```
enum struct Color : unsigned char {
    Red = 254, Green, Blue    // geçersiz! Blue değeri sınırı aşmış
};
```

C++03'teki klasik *enum* türleri *C++11*'de geçmişe doğru uyumu korumak için aynı biçimde bırakılmıştır. Fakat bu eski *enum* türleri de istenirse :: operatörü ile nitelikli olarak kullanılabilir. Örneğin:

```
enum Color {
    Red, Green, Blue
};

//...

Color color1 = Green;             // geçerli
Color color2 = Color::Green;      // geçerli
```

C++11'deki bu yeni *enum* türleri hem sentaks hem de semantik bakımdan C#'taki *enum* türlerine çok benzemektedir.

4. C#'ta enum Türleri

C# standartlarında (*ECMA-334, ISO/IEC 23270*) *enum* türlerine ilişkin şu önemli belirlemeler yapılmıştır:

1. Her *enum* ayrı bir tür belirtir. Her *enum* türünün ilişkin olduğu bir tamsayı türü (*underlying integer type*) vardır. *enum* türünün ilişkin olduğu tamsayı türü *byte*, *sbyte*, *short*, *ushort*, *int*, *uint*, *long* ve *ulong* türlerinden birisi olmak zorundadır. Eğer bildirimde *enum* türünün ilişkin olduğu tamsayı türü belirtilmemişse *default* durum *int* kabul edilmektedir (*ECMA-334, 21.1).*

2. *enum* türleri *enum* bildiriminin dışında *enum* ismi belirtilerek nokta operatörüyle kullanılır (*ECMA-334, 21.3)*.

3. *enum* sabitlerine atanan değerler o *enum* türünün ilişkin olduğu tamsayı türünün sınırları içerisinde kalmak zorundadır (*ECMA-334, 21.3)*.

4. *enum* sabitleri *enum* içerisinde değer atama amacıyla kullanıldığında, onların *enum* türünün ilişkin olduğu tamsayı türünden olduğu kabul edilir. Fakat eğer *enum* sabitleri *enum* bildiriminin dışında kullanılmışsa onların ilgili *enum* türünden oldukları kabul edilmektedir (*ECMA-334, 13.2.2).*

5. Bir *enum* türünden tamsayı, gerçek sayı ya da diğer bir *enum* türüne otomatik (*implicit*) dönüştürme yoktur. Fakat bir *enum* türünden bir tamsayı, gerçek sayı ya da diğer *bir enum* türüne tür dönüştürme operatörü ile dönüştürme yapılabilir. Bu durumda dönüştürmenin sanki *enum* türünün ilişkin olduğu tamsayı türü ile yapıldığı varsayılmaktadır.

6. *E* bir *enum* türünü, *e* bu *enum* türünden bir değeri, *U* da *enum* türünün ilişkin olduğu tamsayı türünü temsil ediyor olsun. *u*'nun da *U* türünden ya da bu türe otomatik (*implicit*) olarak dönüştürülen bir tamsayı türünden olduğunu varsayalım Bu durumda:

a) Bir *enum* türüyle *o enum* türünün ilişkin olduğu tamsayı türüne otomatik olarak dönüştürülebilen bir tamsayı türü toplama işlemine sokulabilir. Sonuç ilgili *enum* türündendir. Yani *e + u* ya da *u + e* işlemi geçerlidir. Sonuç *(E) ((U) e + u)* biçiminde elde edilir (*ECMA-334, 14.7.4).*

b) Bir *enum* türünden o *enum* türünün ilişkin olduğu tamsayı türüne otomatik olarak dönüştürülebilen bir tamsayı türü çıkartılabilir. Sonuç ilgili *enum* türünden olur. Fakat bir tamsayı türünden bir *enum* türü çıkartılamaz. Yani *e - u* işlemi geçerlidir fakat *u - e* işlemi geçerli değildir. *e - u* işlemi *(E) ((U) e - u)* ile eşdeğerdir (*ECMA-334, 14.7.5).*

c) Aynı türden iki *enum* çıkartma işlemine sokulabilir fakat toplama işlemine sokulamaz. Çıkartma işleminden elde edilen değer ilgili *enum* türünün ilişkin olduğu tamsayı türündendir. Yani *e1* ve *e2, E* isimli *enum* türünden olmak üzere, *e1 - e2* ifadesi *(U) e1 - (U) e2* ile eşdeğerdir (*ECMA-334, 14.7.5).*

7) C#'ta *enum* türleri kategori olarak değer türlerine (*value types*) ilişkindir (*ECMA-334, 11.1)*. Tüm *enum* türlerinin *System.Enum* isimli *abstract* sınıftan türetilmiş olduğu varsayılmaktadır (*ECMA-334, 21.4)*.

Şimdi yukarıdaki belirlemeleri bazı örneklerle açıklayalım. Öncelikle, *C#'ta* da her *enum* türünün ilişkin olduğu bir tamsayı türü vardır. Bu tür *enum* sentaksında *enum* isminden sonra *':'* atomu ile belirlenmektedir. Örneğin:

```
enum Color : byte
{
    Red, Green, Blue
}
```

Eğer *enum* türünün ilişkin olduğu tamsayı türü belirtilmezse *default* olarak *int* varsayılmaktadır. *C#*'ta *enum* türlerinden tamsayı ve gerçek sayı türlerine, tamsayı ve gerçek sayı türlerinden de *enum* türlerine otomatik (*implicit*) dönüştürme yoktur fakat tür dönüştürme operatörü ile (*explicit*) dönüştürme yapılabilmektedir. Örneğin:

```
Color c = 1;                // geçersiz!
Color c = Color.Green;      // geçerli

int i;

i = c;             // geçersiz!
i = (int) c;       // geçerli
```

Bir *enum* türünden değişken için o *enum* türünün ilişkin olduğu tamsayı türü kadar yer ayrılmaktadır. *enum* sabitleri kendi *enum* türlerinin ilişkin olduğu tamsayı türünün dışında değer alamazlar. Örneğin:

```
enum Direction : byte
{
    Up = 253, Right, Down, Left     // geçersiz! Left = 256 değerinde
}
```

C#'ta *enum* türleri *C* ve C++'ta olduğu gibi işlem öncesinde tamsayı türlerine ya da gerçek sayı türlerine otomatik olarak dönüştürülmezler. Fakat bir *enum* türleriyle tamsayı türleri toplama ve çıkartma işlemine sokulabilirler. Bu durumda sonuç ilgili *enum* türünden olur. Ayrıca, bir *enum* türünden değişken içerisindeki değerin o *enum* türünün sabitlerine ilişkin bir değer olması da zorunlu değildir. Örneğin:

```
enum Color
{
    Red, Green, Blue
}

//...

Color c = Color.Green;
Color result;

result = c + 1;        // geçerli, result içerisinde Color.Blue değeri var
//..
result = c - 1;        // geçerli, result içerisinde Color.Red değeri var
//...
result = c + 100;      // geçerli, result içeriisnde 101 var
```

Aynı türden iki *enum* toplanamaz, fakat çıkartılabilir. Çıkartma işleminden elde edilen değer *enum* türünün ilişkin olduğu tamsayı türündendir. Örneğin:

```
Color c1 = Color.Blue;
Color c2 = Color.Red;
int result;

result = c1 - c2;       // geçerli, result = 2
```

enum sabitlerine değer atanırken önceki ya da sonraki *enum* sabitleri kullanılabilir. Fakat döngüsel bir durum oluşturulamaz. Örneğin aşağıdaki gibi bir bildirim geeçerlidir:

```
enum Day
{
    Sunday = 10, Monday = Sunday + 10, Tuesday = Friday -10,
    Wednesday, Thursday, Friday = 50, Saturday
}
```

C#'ta tüm *enum* türlerinin *System* isim alanı içerisindeki *Enum* isimli *abstract* sınıftan türetildiği varsayılmaktadır. *System.Enum* sınıfı *System.ValueType* sınıfından, *System.ValueType* sınıfı da *System.Object* sınıfından türetilmiş durumdadır. Böylece *C#*'ta biz herhangi bir *enum* türünü *System.Enum*, *System.ValueType* ya da *System.Object* türüne doğrudan atayabiliriz. Bu durumda kutulama dönüştürmesi (*boxing conversion*) gerçekleşecek ve *enum* değişkeninin *heap*'te bir kopyası çıkartılacaktır. Örneğin:

```
Color c = Color.Red;
object obj;

obj = c;            // geçerli, obj heap'te kopyası çıkartılmış
                    // olan c'yi gösteriyor
```

Color nesnesini yeniden ilgili *enum* türüne dönüştürebiliriz. Bu durumda kutuyu açma dönüştürmesi (*unboxing conversion*) gerçekleşecek ve *heap*'teki *Color* nesnesinin *stack*'te geçici bir kopyası oluşturulacaktır:

```
Color other;

other = (Color) obj;     // kutuyu açma dönüştürmesi (unboxing conversion)
```

.NET ortamında ara kodlu bir çalışma modeli söz konusu olduğundan *metadata* bilgilerinden hareketle bir programın türleri hakkında ayrıntılı bilgiler edinilebilmektedir. İşte *System.Enum* sınıfının da bu *metadata* bilgilerini kullanarak işlemlerini yapan çeşitli faydalı metotları vardır. Örneğin, *Enum* sınıfının *GetName* isimli *static* metodu *enum* türünün *Type* nesne referansını ve o *enum* türünden bir değeri parametre olarak alır; bize o değere sahip *enum* sabitinin ismini verir. Metodun parametrik yapısı şöyledir:

```
string GetName(Type t, object val)
```

GetNames metodu da benzer biçimde bir *enum* türüne ilişkin tüm sabit isimlerini vermektedir:

```
public static string[] GetNames(Type enumType)
```

Bu işlemin tam tersini yapan *GetValues* metodu da vardır. Bu metot *enum* türünün *Type* nesne referansını alır ve o *enum* türünün tüm sabitlerini o *enum* türünden bir dizi olarak verir:

```
public static Array GetValues(Type enumType)
```

5. Java'da Enum Türleri

Java'daki *enum* türleri *C*, *C++* ve *C#*'tan oldukça farklıdır. Bu dilde *enum* türleri bir sınıf gibi ele alınmaktadır. Bu nedenle *Java*'daki *enum* türleri başlangıç metotlarına (*constructors*), veri elemanlarına (*fields*) ve metotlara sahip olabilirler. Örneğin:

```
public enum Color
{
    Red, Green, Blue;

    private int a;
    private int b;
    private static int c;

    private Color()
    {
        a = 10;
        b = 20;
    }

    public void disp()
    {
        System.out.printf("a = %d b = %d c = %d\n", a, b, c);
    }

    public static int getC()
    {
        return c;
    }

    public static void setC(int c)
    {
        Color.c = c;
    }

    //...
}
```

```
//...

Color c = Color.Red;

Color.setC(30);

c.disp();
```

Java'daki *enum* bildiriminde önce *enum* sabitlerinin belirtilmesi gerekir. Eğer *enum* sabitlerini diğer sınıfın diğer eleman bildirimleri izleyecekse *enum* sabit listesinin ';' atomu ile sonlandırılması zorunludur.

Enum sabitleri ilgili *enum* türündendir ve bu türden birer referans belirtirler. Böylece biz bir *enum* sabitini aynı türden bir *enum* referansına atayabiliriz. Örneğin:

```
Color c = Color.Red;
```

enum sabitleri sınıf ilkdeğerlemesi yapılırken (*class initialization*) yaratılırlar. Yaratılmaları sırasında *enum* türünün uygun başlangıç metodu (*constructor*) çağrılmaktadır. *enum* sabitleri dışında *new* operatörü ile o *enum* türünden başka nesneler yaratılamaz. *enum* sabitlerinin yaratılması sırasında hangi başlangıç metodunun çağrılacağı *enum* sabitinden sonra parantezler içerisinde belirtilen argüman listesine bağlıdır. Eğer *enum* sabitinden sonra parantez açılmazsa yaratım için *enum* türünün parametresiz başlangıç metodu çağrılır. Örneğin:

```
enum Color
{
    Red(10), Green, Blue(20);

    Color()
    {
        //...
    }

    Color(int val)
    {
        //...
    }
    //...
}
```

Burada *Red* ve *Blue* için *int* parametreli başlangıç metodu, *Green* için parametresiz başlangıç metodu çağrılacaktır.

Yukarıda da belirttiğimiz gibi, *Java*'da *enum* türlerine ilişkin *enum* sabitleri ilgili *enum* türünden sınıf referansları belirtmektedir. Örneğin:

```
enum Direction
{
    Up, Right, Down, Left
}
```

Burada *Up*, *Right*, *Down* ve *Left* sabitleri *Direction* isimli *enum* sınıfı türünden nesne referansları belirtir. Bu nedenle iki *enum* türünün karşılaştırılması semantik olarak iki referansın karşılaştırılması işlemi ile aynıdır. Dolayısıyla *Java*'da biz aynı türden iki *enum* değerini == ve !=

operatörleriyle karşılaştırma işlemine sokabiliriz fakat >, <, >= ve <= operatörleriyle karşılaştırma işlemlerine sokamayız. *Java*'da *enum* türleri *switch* işlemine sokulabilirler. Örneğin:

```
public static void Move(Direction d)
{
    switch (d)
    {
        case Up:
            //...
            break;
        case Right:
            //...
            break;
        case Down:
            //...
            break;
        case Left:
            //...
            break;
   }
}
```

---

[1] Bu durum eleştirilebilir. *enum* türlerinden gerçek sayı türlerine otomatik (*implicit*) dönüştürme olduğuna göre bunun tersinin de tür dönüştürme operatörü ile yapılabilmesi beklenirdi.

Kaynaklar

1. ANSI/ISO 9899:1990 (1990). *American National Standard for Programming Languages - C*. New York: American National Standard Institute.

2. ISO/IEC 9899:1990 (1999). *International Standard - Programming Languages - C, Second Edition.*

3. ISO/IEC 14882 (2003). *International Standard - Programming Languages - C++,*, *Second Edition.*

4. ISO/IEC 14882 (2011). *International Standard , Information Technology - Programming Languages - C++, Third Edition.*

5. ECMA International (2006). *Standard ECMA-334, C# Language Specification, Forth Edition.*

6. Gosling J., Joy B., Steele G., Bracha G. (2005). *The Java Language Specification, Third Edition*. Addison-Wesley