

Masaüstü Uygulama Geliştirme

Hafta-3

Öğr. Gör. Zafer SERİN

Değişken İsimlendirme Kuralları

- Genel olarak programlama dillerinde değişkenlere isim verirken uygulanan bazı yöntemler vardır. Bunlar: snake_case, PascalCase ve camelCase olarak adlandırılır.

- Örneğin;

`int birinci_deger; // snake_case kullanımı`

`int BirinciDeger; // PascalCase kullanımı`

`int birinciDeger; // camelCase kullanımı`

- Bu tarz değişken isimlendirmeleri zorunlu olmamakla birlikte bir düzen ve kod anlaşılabilirliği sağladığı için önerilmektedir.

@ operatörü kullanımı

- Değişken isimlerinde programatik keyword kullanılamaz! Eğer ki bir değişken isminde programatik olarak kullanılan bir keywordü vermek istiyorsanız eğer bunu @ operatörü ile kullanmanız gerekir.
- Örneğin `int @static = 5;`

Tanımlanmış Değişkene Değer Atama

- C# dilinde tanımlanmış bir değişkene doğrudan tanımlandığı satırda veya sadece değişken tanımlaması yapılarak, kodun geri kalan kısımlarında atama yapılabilir.

- Örneğin;

```
int birinci_deger = 5; // Değişkene tanımlandığı yerde atama
```

```
int birinci_deger;
```

```
birinci_değer = 15;
```

- Buradaki '=' operatörü matematiksel eşitlik anlamında değil atama operatörü olarak kullanılmıştır.

Tanımlanmış Değişkene Değer Atama

- Değişkene değer atanırken = işaretinin sol tarafına değişkenin kendisi gelirken sağ tarafına değeri gelecektir.

```
int birinci_deger = 15;
```

```
int ikinci_deger = 25;
```

```
int ucuncu_deger = 35;
```

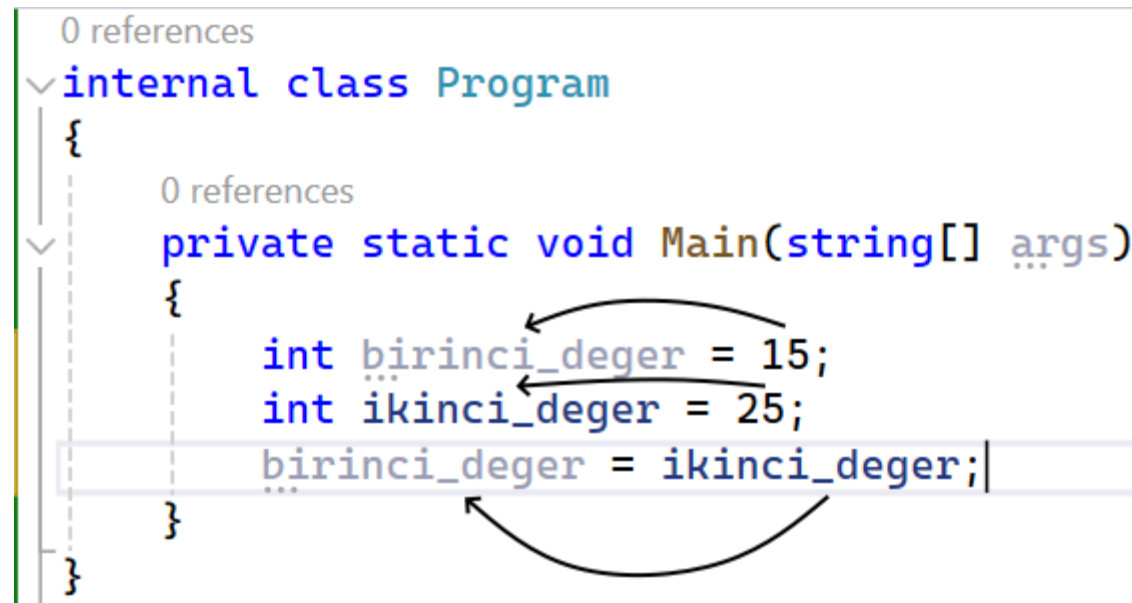
```
ikinci_deger = ucuncu_deger;
```

- Bu durumda ikinci_deger kısmına değişkenin kendisi gelirken ucuncu_deger kısmına ucuncu_degerin kendisi değil değeri(35) gelecektir.

Tanımlanmış Değişkene Değer Atama

- Değişken tanımlaması ve ataması yapıldığında daima sağdaki değer soldaki değişkene atanır.

```
0 references
internal class Program
{
    0 references
    private static void Main(string[] args)
    {
        int birinci_deger = 15;
        int ikinci_deger = 25;
        birinci_deger = ikinci_deger;
    }
}
```



Kurallar

- Bir değişkene atanan en son değer geçerlidir.

```
int birinci_deger = 15;
```

```
birinci_deger = 25;
```

```
birinci_deger = 57;
```

- Bu durumda birinci_deger değişkenine son atanan değer 57'dir.
- Tanımlanmış olan değişkene türüne uygun bir değer atanmalıdır.

```
int birinci_deger = "selam";
```

- Bu durumda hata ile karşılaşılır. Çünkü int bir değişken tanımlanmış ama bu değişkene string bir değer atanmaya çalışılmaktadır.

Metinsel Değerler

- Temel olarak string değişken tipi ile ifade edilir.
- Çift tırnak içerisinde değeri yazılır.
- Bir sayısal ifade metinsel olarak tutuluyorsa yazılım açısından metinsel olarak ifade edilir.
- Örneğin;

```
string isim = "Zafer";
```

```
string ifade = "Zafer 123";
```

```
string metinsel = "21";
```

```
string karisik = "Zafer123*!";
```


Karaktersel Değerler

- Temel olarak char değişken tipi ile ifade edilir.
- Tek tırnak içerisinde değeri yazılır.
- Tek bir karakteri tutarlar.
- Örneğin;

```
char a_karakter = 'a';
```

```
char yildiz_karakter = '*';
```

```
char unlem_karakter = '!';
```

```
char son_karakter = 'n';
```

Mantıksal Değerler

- Temel olarak bool değişken tipi ile ifade edilir.
- true veya false değer alabilir.
- Var-yok, doğru-yanlış, 1-0 gibi düşünülebilir.
- Atama yapılırken herhangi bir tırnak işareti kullanılmaz!
- Örneğin;

```
bool giris_yapildimi = true;
```

```
bool sart_saglandimi = false;
```

```
bool islem_tamamlandimi = true;
```

Sayısal Değerler

- Temel olarak int, float, double ve decimal gibi tipler ile ifade edilirler.
- Tamsayılarda varsayılan değer int, ondalıklı sayılarda double'dir.
- float değişken tanımlanırken değer sonuna f veya F, double için d veya D, decimal için m veya M eklenir. Bunlara kapsam dışı atama yapılamaz!
- Örneğin;

```
int sayi1 = 15;
```

```
float sayi2 = 25.7f;
```

```
double sayi3 = 21.65D; // double sayi3 = 21.65;
```

```
decimal sayi4 = 2156.543m;
```

Tuple ile Değer Atama

- Tek bir syntax ile birden fazla değişkene değer atamak için kullanılır.
- Örneğin;

```
(int yas, string isim) tuple1 = (25, "Zafer");
```

```
(float boy, char karakter) tuple2;
```

```
tuple2 = (1.70f, 'z');
```

```
(int sayi1, string okul) tuple3;
```

```
tuple3.sayi1 = 30;
```

```
tuple3.okul = "PMYO";
```

Literals Düzenlemeler

- C# 7.0 ile gelen bir düzenleme ile büyük sayısal değerlerde '_' ile basamak ayrımı yapılabilmesi sağlanmıştır.
- Örneğin;

```
int sayi1 = 1_000_000; // int sayi1 = 1000000;
```

Varsayılan Değer Atama

- Bir değişken class içerisinde tanımlanırsa varsayılan bir değere sahip olacaktır. Bunlar; string için null, char için \0, sayısal ifadelerde 0 ve bool için ise false olur.
- Main içerisindeki bir değişkene ise varsayılan değer doğrudan atanmaz! İlgili değer elle verilmesi gerekir. Bir değişkene varsayılan değeri vermek için default sözcüğü kullanılır.

```
int sayi1 = default(int); // int sayi1 = default; c# 7.1 ile gelen özellik  
bool giris_yapildimi = default(bool);  
char karakter = default(char);
```

Tanımlanmış Değişkenin Değerini Okuma

- Bir değişkenin değerini elde edebilmek için değişkenin adından faydalanmaktayız. Bir değişkenin adı assign(=) operatörünün sağında yahut metotların parametrelerinde çağrılıyorsa ilgili değişkenin değeri gönderilir.

- Örneğin;

```
int sayi1 = 25;
```

```
Console.WriteLine(sayi1);
```

```
// 4 sayılı örnek
```

Değişkenlerin Faaliyet Alanları(Scope)

- Faaliyet alanı değişken ve fonksiyonların erişilebilirlik sınırlarını belirleyen alandır.
- Tanımlamalarda ve algoritmik çalışmalarda karışıklığı önleyen bir sınır çizer.
- C#'ta faaliyet alanları süslü parantezler ile belirlenir.
- Bir değişken sade ve sadece tanımlandığı scope içerisinde erişilebilir ve kullanılabilir.
- Süslü parantez ile istenilen yerde scope oluşturulabilir.

Değişmeyenler(Sabitler)

- Sabitler; değişmeyen değerleri tutmak için oluşturulmuş yapılardır.
- Süreçte var olan değeri değiştirilemez, değiştirilmeye çalışıldığı takdirde compiler tarafından hata verilir.
- C#'ta const anantar sözcüğü ile kullanılır ve açılımı constant'tır. Değişmeyendir. Prototip olarak değişkenlere çok benzer lakin davranışsal olarak değeri bir daha değiştirilemez. Özünde static yapılanmadır.
- Static yapılanma uygulama bazlı veri depolayabildiğimiz bellekte bir alandır. Static değişkenlerin const tan farkı değerlerinin değişebilmesidir.

Değişmeyenler(Sabitler)

- Bir diğer sabit türü readonly olarak düşünülebilir. Bu sadece okunabilir değişkenler tanımlamak için kullanılır.
- Const'tan farkı sadece tanımlandığı yerde değil, ayrıca constructor içerisinde de değeri atanabilir. Dependency Injection deseninde çok sık tercih edilir.
- Ayrıca static değildirler.
- Const'lar tanımlanır tanımlanmaz değerini almak zorundadır.
- Bir const tanımlandığında Stack'te ilgili türde bir alan tahsis edilecektir ve ilk atanan değer dışında bir daha değer kabul etmeyecektir.

Değişmeyenler(Sabitler)

- Const'lar değiştirilemez lakin istenildiği kadar okunabilir.
- Const değişkenlere tanımlama aşamasında değer verilmelidir.
- `const degisken_tipi degisken_adi;` şeklinde c#'ta tanımlanabilir.

Değişken Atama Varyasyonları

- Aynı türden değişkenler tek bir satırda atanabilir. Örneğin

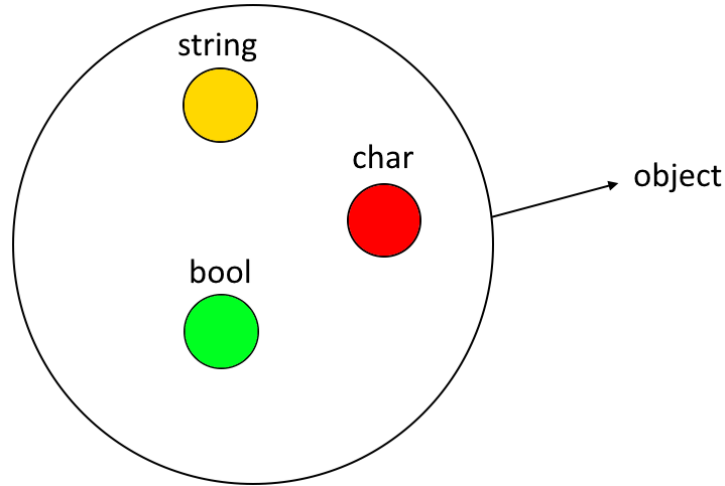
```
int sayi1 = 5;
```

```
int sayi2 = 10;
```

```
int sayi3 = 5, sayi4 = 10;
```

Object Türü

- Object türü tüm türleri karşılayabilen bir türdür. Boxing yöntemi ile değişken kutulanır ve bir object tür haline getirilir. Unboxing ile ise tekrar dışarı alınarak gerçek değer elde edilir.



var ifadesi

- Tutulacak değerin türüne uygun bir değişken tanımlayabilmek için kullanılan bir keywordtür.
- var keywordü kendisine atanan değerin türüne bürünür.
- Bir değişken türü değildir.
- Asıl amacı farklı diller arasında uyum sağlamaktır.
- Örneğin;

```
var sayi1 = 15;
```

```
var isim = "Zafer";
```

```
var boy = 1.70f;
```

dynamic ifadesi

- var ile benzer bir kullanıma sahip olmakla birlikte var derleme aşamasında değerin türüne bürünürken, dynamic ise runtime da (çalışma zamanında) değerin türüne bürünür.
- GetType() metodu ile değişkenlerin türü öğrenilebilir.
- Çalışma zamanında uzaktan gelen verileri karşılamak için kullanılır.

Tür Dönüşümleri(Type Conversion)

- Yazılım sürecinde elimizdeki değerlerin türlerini değiştirebilmekteyiz.
- Tür dönüşümleri elimizdeki verinin mahiyetindeki türe uygun işlemlere tabi tutulması için uygulanabilir.
- Örneğin elimizde string tipte "123" değeri varsa ve biz bunu 2 ile çarpmak istersek sorun yaşarız. Bu gibi durumlarda tür dönüşümü kullanılabilir.
- Ayrıca farklı servislerden gelen değerleri uygun türlere dönüştürmek içinde kullanılır.

Tür Dönüşümleri(Type Conversion)

- Tür dönüşümlerinde amaç türü dönüştürmektir. Yani elimizdeki veriye uygun bir türe geçiş yapmaktır. Elimizdeki veriyi uygun olmayan bir türe dönüştürmeye çalışırsak bu mümkün değildir. Hata verecektir. Amaç veriyi değiştirmek değil dönüştürmektir.
- Örneğin "Zafer" şeklindeki bir string ifadeyi inte çeviremeyiz.

Metinsel İfadelerin Diğer İfadelere Dönüştürülmesi

- Burada kullanılan 2 yöntem vardır. Bunlar Parse ve Convert fonksiyonlarıdır. Parse yöntemi sadece string dataları hedef türe dönüştürürken kullanılır.
- Parse kullanımı şu şekildedir: `type.Parse()`; burada type ile ifade edilen dönüştürülecek tiptir. int, bool vb.

```
string deger = "123";
```

```
int deger_int = int.Parse(deger);
```

Metinsel İfadelerin Diğer İfadelere Dönüştürülmesi

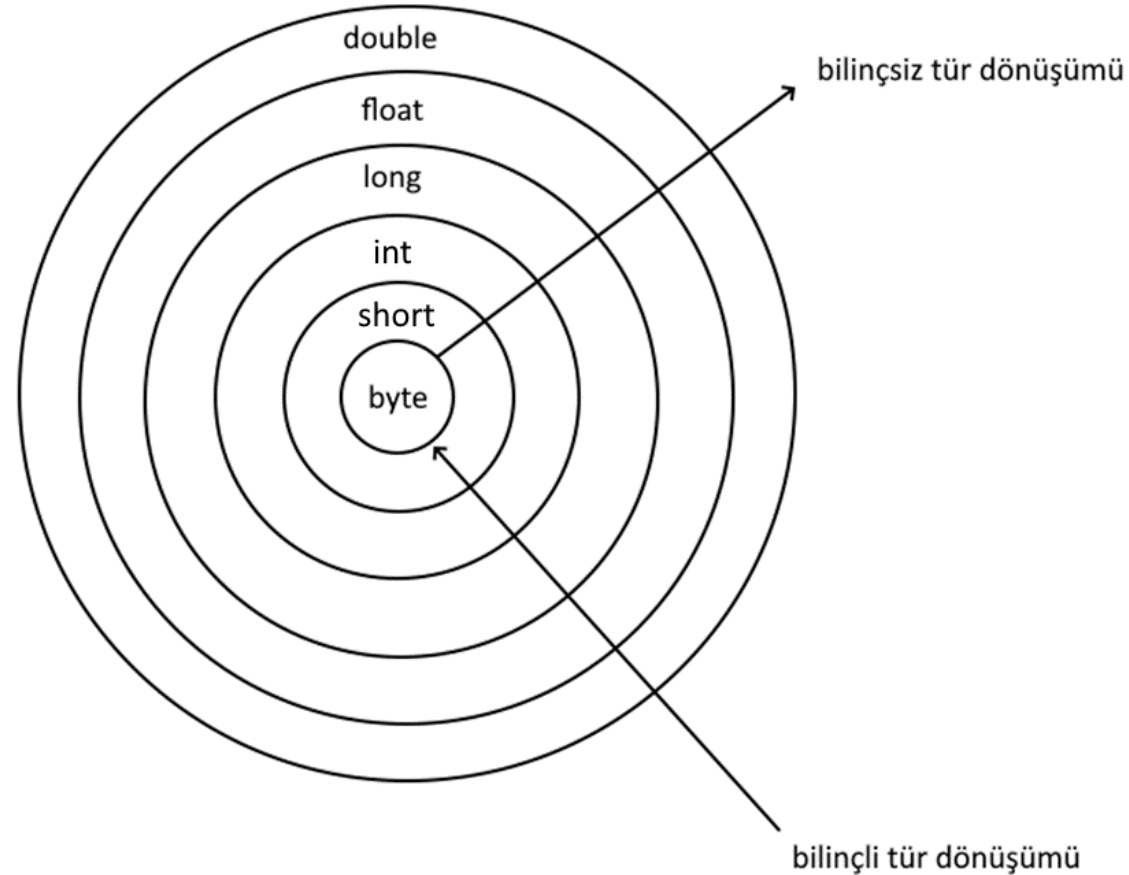
- Convert yöntemi sadece string için değil tüm tür dönüşümleri için kullanılabilir; ancak yine tip uyumluluğu olmalıdır.

```
string giris_yapildimi = "TruE";  
Convert.ToBoolean(giris_yapildimi);
```

Sayısal İfadelerin Kendi Arasında Dönüştürülmesi

- Sayısal ifadelerin dönüşümü bilinçli ve bilinçsiz tür dönüşümü olarak 2'ye ayrılır. Bilinçsiz tür dönüşümü kendinden daha geniş değerler tutabilen sayısal değişken tiplerine dönüşümde kullanılırken, bilinçli tür dönüşümü kendinden daha dar değerler tutabilen sayısal değişken tiplerine dönüşümde kullanılır.
- Bilinçli tür dönüşümünde veri kaybı olabilir.

Sayısal İfadelerin Kendi Arasında Dönüştürülmesi



Sayısal İfadelerin Kendi Arasında Dönüştürülmesi

```
byte sayi1 = 25;
```

```
int sayi2 = sayi1;
```

```
double sayi3 = 2500;
```

```
float sayi4 = (int)sayi3;
```

checked ve unchecked Yapıları

- Bilinçli tür dönüşümü sırasında veri kaybı olabilecek bir durum varsa bunu kontrol etmek için checked bloğu kullanılır ve veri kaybı olursa kullanıcıyı uyarır. unchecked bloğunda ise bu kontrol yapılmaz. Varsayılan durum unchecked'tir.

```
checked{
```

```
    double sayi1 = 5000d;
```

```
    byte sayi2 = (byte)sayi1;
```

```
    Console.WriteLine(sayi2);
```

```
}
```

Mantıksal Türlerin Sayısal Türlere Dönüşümü

- Mantıksal ifadeler sayısal türlere dönüştürülürken true değeri 1'e ve false değeri ise 0'a atanır.

`bool deger = true;`

`int sayi = Convert.ToInt32(deger);`

- Tam tersi durumda bir sayısal değer mantıksal değere dönüştürülecekse burada bir istisna vardır. 0 değeri false olurken 0 dışındaki tüm değerler true olarak dönüşecektir.

Sayısal Türlerin Mantıksal Türlere Dönüşümü

- Bir sayısal değer mantıksal değere dönüştürülecekse burada bir istisna vardır. 0 değeri false olurken 0 dışındaki tüm değerler true olarak dönüşecektir.

Karakter Türlerin Sayısal Türlere Dönüşümü

- Karakter(char) olarak tanımlanmış değişkenler sayısal ifadelere dönüştürülürken onların ascii tablo karşılıkları elde edilir.

```
char a_karakter = 'a';
```

```
int b = Convert.ToInt32(a_karakter);
```

Sayısal Türlerin Karakter Türlere Dönüşümü

- Karakter(char) olarak tanımlanmış değişkenler sayısal ifadelere dönüştürülürken onların ascii tablo karşılıkları elde edilir.

```
int sayi = 97;
```

```
char a_karakteri = Convert.ToChar(sayi);
```