Masaüstü Uygulama Geliştirme

Hafta-5

Öğr. Gör. Zafer SERİN

Akış Kontrol Mekanizmaları

- Akış kontrol mekanizmaları, kodun akış sürecinde belirli şartlara göre farklı yönlendirmeleri yapmamızı ve farklı algoritmaları/kodları/yapılanmaları çalıştırmamızı sağlayan mekanizmalardır.
- if-else ve switch-case yapılanmaları olarak kullanılır. Bunlar arasında teknik olarak fark olmamasına karşın kullanım açısından bir fark yoktur.
- Algoritmalarda çok sık kullanılır ve oldukça önemli mekanizmalardır.

Switch-Case Yapısı

- Switch-case, kodun akışında belirli bir şarta göre yönlendirme yapmamızı sağlayan yapılanmadır.
- Sadece eşitlik durumu kontrol edilecekse o zaman switch kullanılabilir.

Switch-Case Yapısı

```
switch(kontrol_edilecek_deger)
{
        case deger1:
        // kontrol_edilecek_deger deger1'e eşitse yapılacak işlemler
        break;
        case deger2:
        // kontrol_edilecek_deger deger2'ye eşitse yapılacak işlemler
        break;
        default:
        // kontrol_edilecek_değer hiçbir degere eşit değilse yapılacak işlemlerdir. Default kullanımı zorunlu değildir!
        break;
}
```

Switch-Case Yapısı

- Yalnızca bir case durumu çalışacaktır. Bir şart sağlanırsa diğerlerine bakılmaz!
- Switch parantezinde kontrol_edilecek_deger bir değişken yahut statik bir değer olabilirken case bloklarındaki değerler kesinlikle statik olmak zorundadır. Caselerdeki değerler değişkenlerden alınamaz!
- Case bloklarının sıralaması ve defaultun yerleştirildiği yer önemli değildir.

Switch-Case Yapısı When Kullanımı

- Switch yapılanmasında eşitlik ile ilgili ek bir şart daha kontrol edilmek istenirse when yapısı kullanılır.

```
var sayi = 25;
switch (sayi)
{
    case 25 when (true):
        Console.WriteLine("Görüşürüz");
        break;
}
```

if Yapısı

- İf yapılanması, elimizdeki bir değerin eşitlik durumuda dahil tüm karşılaştırmaları yapmamızı sağlayan ve sonuca göre akışı yönlendirmemizi sağlayan bir yapılanmadır. Else kullanımı zorunlu değildir!

```
if(sart)
{

}
```

if-else Yapısı

- Bir şartın sağlandığı durumda bir kod bloğunu çalıştırmak için if yapısını kullanıyoruz. Eğer ilgili şart sağlanmaz ise çalışmasını istediğimiz bir kod bloğu var ise bunu da else yapısı ile belirtiyoruz.

```
if(sart){
// şart sağlanırsa yapılacaklar
}
else{
// şart sağlanmazsa yapılacaklar
}
```

if-else Yapısı

- Hem if bloğunda hem de else bloğunda çalışacak bir kod varsa bu kod mutlaka ilgili blokların dışına yazılmalıdır.(Kendimizi tekrar etmiyoruz!)
- if bloğu içerisinde tek satırlık bir işlem yapılacaksa süslü parantez kullanılmadan da işlem yapılabilir!(Scopesuz if yapısı)

if-else if-else Yapısı

- Sadece iki durumun değil farklı durumlarında kontrol edilmesi durumunda bu yapı kullanılabilir. Birden fazla şartı kontrol etmemizi sağlar. İf yapılanmalarından herhangi birisi sağlandıysa diğer ifler kontrol edilmeden geçilecektir.

```
if(ilk_sart){// ilk_sart sağlanırsa yapılacaklar}
else if(ikinci_sart){// ikinci_sart sağlanırsa yapılacaklar}
else if(ucuncu_sart){// ucuncu_sart sağlanırsa yapılacaklar}
else{// hiçbir şart sağlanmazsa yapılacaklar}
```

Type Pattern

- Object içerisindeki bir tipin belirlenmesinde kullanılan is operatörünün desenleştirilmiş halidir.
- is ile belirlenen türün direkt dönüşümünü sağlar.
- var anahtar sözcüğü ile kullanılabilir.

```
object sayi = 25;
if(sayi is int sayi_i)
{
    Console.WriteLine("Selam");
}
```

Constant Pattern

- Elimizdeki veriyi sabit bir değer ile karşılaştırabilmemizi sağlar.

```
object sayi = 25;
if(sayi is 25)
{
    Console.WriteLine("Selam");
}
```

Örnek1

- Girilen 3 sayı içerisinden en büyüğünü bulan kodu yazınız.

Örnek2

- Öğrencinin girdiği nota göre harf notunu hesaplayan kodu yazınız.

a) Başarı notu	Mutlak sistem karşılığı
AA.....	93,01-100,00
AB.....	83,67-93,00
BA.....	76,67-83,66
BB.....	69,67-76,66
BC.....	60,34-69,66
CB.....	56,61-60,33
CC.....	53,33-56,60
DC.....	49,00-53,32
DD.....	45,00-48,99
FF.....	.....44,99 ve aşağısı.

Döngüler

- Tekrar eden kodları bir koşula bağlı olarak belirli sayıda tekrarlayabilen yapılara döngü diyoruz.
- Koşul sağlandığı sürece ilgili döngü içindeki kodları tekrar eder.

for Döngüsü

- Aşağıda oluşturulan for döngüsü int i = 0; kodu ile bir i değişkeni tanımlamış ve buna 0 değerini atamıştır. Daha sonra bu değerin(0’ ın) 10’dan küçük olup olmadığını kontrol etmiştir. 10’dan küçük olduğu için ekrana Selam yazdırmış ve sonra i’yi 1 artırmıştır. Daha sonra tekrar bu değerin (1’in) 10’dan küçük olup olmadığını kontrol etmiş ve 10’dan küçük olduğu için tekrar ekrana Selam yazdırmıştır. Bu şekilde döngü devam eder.

```
for (int i = 0; i < 10; i++){Console.WriteLine("Selam");}
```

while Döngüsü

- Aşağıda oluşturulan while döngüsü bir önceki örnekteki for döngüsü ile aynı işi yapar. Burada i değeri daha önce tanımlanmıştır ve i değeri 10'dan küçük olduğu sürece while scopeu({}) içerisindeki işlemler yapılır. Döngü içinde i'nin artırıldığına(i++) dikkat edilmelidir. Yoksa sonsuz döngü meydana gelir.

```
int i = 0;
while(i < 10){
    Console.WriteLine("Selam");
    i++;}
```

do while Döngüsü

- do while döngüsü while'dan farklı olarak şart geçerli olmasa bile ilgili bloktaki kodu 1 kez çalıştırmayı sağlar.

```
int i = 0;
do
{
    Console.WriteLine("Selam");
    i++;
} while (i > 1000);
```

break ve continue anahtar sözcükleri

- Bu anahtar sözcükler döngüye müdahale etmemizi sağlar.
- break anahtar sözcüğü döngüyü sonlandırmayı sağlarken; continue anahtar sözcüğü yazıldığı yerdeki döngü iterasyonunu sonlandırıp bir sonraki iterasyondan devam edebilmeyi sağlar.

Örnek3

- 1’den 100’e kadar olan sayıları yazdırınız.

Örnek4

- 150 ile 450 arasındaki 15’in katı olan sayıları bularak ekrana yazdırınız.

Diziler(Arrays)

- Diziler içerisinde birden fazla aynı türde değer tutabilen yapılardır.
- Prosedürel programlamanın temel veri kümeleridir. Yani yazılımsal boyutta, yazılım adına RAM'de birden fazla değeri tek bir değişken altında bir veri kümesi halinde tutabilirler.
- Diziler veri kümeleri oldukları için, içlerindeki birden fazla veri üzerinde kümesel işlemler yapmamızı sağlayabilirler.
- Diziler, prosedürel programlamanın temel yapıları oldukları için daha gelişmiş yapılar olan koleksiyonlarında temelini oluştururlar ve gelişmelerine katkıda bulunurlar.

Diziler(Arrays)

- Diziler referans türlü değerlerdir. Yani nesnel yapılardır. Özlerinde classtırlar.
- Yapısal olarak RAM'de Heap'te tutulurlar.
- Dizi içerisine her türlü değer koyulabilir.(Değer türlü-referans türlü)
- Diziler içerisine koyulan değerler işlevsel olarak aynı mahiyette olmalıdır.
- Diziler içerisine eleman/değer eklendikçe bunları rastgele bir şekilde depolamaz! Düzenli bir şekilde sıralı depolayacaktır.
- Dizilerde eklenen elemanlar index(indis) ismini verdiğimiz numaralarla ardışık bir şekilde numaralandırılırlar.

Diziler(Arrays)

- İndex(indis) numarası her bir elemanı verilen ardışık sayıdır ve 0'dan başlayıp n-1'de sonlanır. n burada dizinin eleman sayısıdır.

type[] isim = new type[eleman_sayisi];

- Dizi tanımlama sürecinde eleman sayısı mecburi girilmek zorundadır. Yani dizide çalışacak değer adeti başta bildirilmelidir. Bu çok önemli bir sınırlılıktır.
- Bir dizi tanımlaması yapıldığı an bellekte o diziyi kullansakta kullanmasakta verilen eleman sayısı kadar alan tahsisinde bulunulur.

Diziler(Arrays)

- Kullanılmadığı halde diziler direkt olarak bellekten belirtilen eleman sayısı kadar alan tahsisinde bulunduğu için bu bir sınırlılıktır.
- Diziler alan tahsisi yapıldığında ilgili alanlara türüne uygun default(varsayılan) değerleri atarlar.

Diziler’de Değer Atama ve Değer Okuma

- Diziler’de değer atama ve okuma işlemleri index’ler üzerinden sağlanır.

```
int[] sayilar = new int[10];
sayilar[3] = 5;
Console.WriteLine(sayilar[3]);
```

- Diziler’de değer atama ve değer okuma işlemleri yapılırken dizinin sınırı aşılmamalıdır!
- Diziler’de değer ataması yapılırken sıralamayı göz önünde tutmak zorunda değiliz.

Diziler’de Değer Atama ve Değer Okuma

- Diziler’de eleman sayısı başlangıçta belirlendiği için duruma göre fazladan değer atayamayız. Bu bir sınırlılıktır.