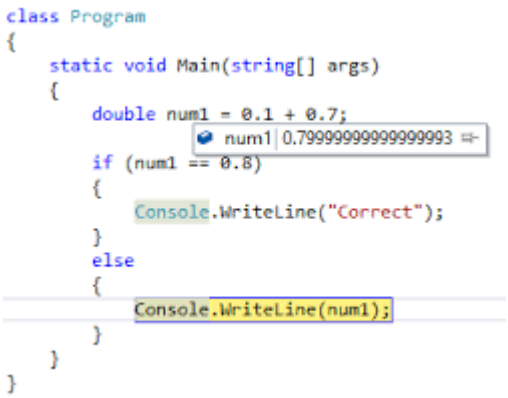
Programlama Hassasiyet İpucu: Noktalı Sayı Kullanımı



Noktalı sayı kullanılırken özellikle eşitlik karşılaştırmalarında kurguladığımız senaryo gereği iki sayının birbirine eşit olma durumu için bir tolerans aralığı seçilmelidir. Aşağıda örnek bir senaryo üzerinde ne demek istediğimizi anlatmaya çalışalım.

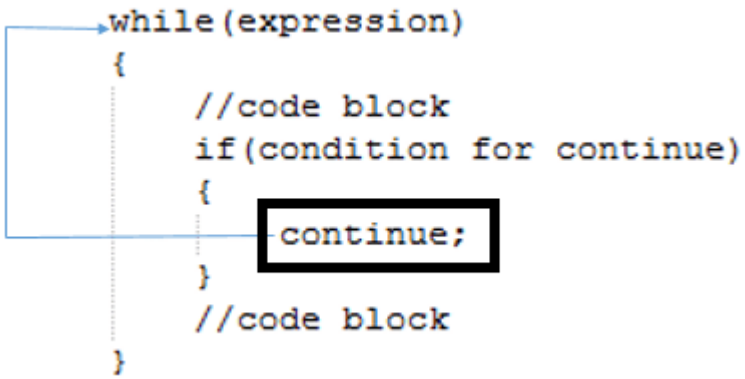
Örneğin demir çelik endüstrisi için bir otomasyon geliştiriyorsunuz ve birim ağırlık olarak kg kullanıyorsunuz. Böyle bir senaryoda 5000 kg ve 5000,005 kg aslında birbirine eşit kabul edilebilir bir aralıktır. Ama programlamada 5000 kg içerisinde aradaki 5 gramlık farktan ötürü eşitlik şartı sağlanmamaktadır ve eşitlik durumunda çalışması gereken kodlar çalışmayacaktır. Bunun için direk if(sayi1 == sayi2) şeklinde bir karşılaştırma değil de aşağıdaki gibi bir tolerans payı ile karşılaştırma yapılmalıdır.

```
const double tolerans = 0.005;
double sayi1 = 5000;
double sayi2 = 5000.005;
```

```
if (Math.Abs(sayi2 - sayi2) <= tolerans)
{
    // eşitlik durumu için çalışacak kodlar
}
```

Kodumuzda if şartı içerisinde aradaki toleransı (+) ve (-) durumlarının ikisinde de sağlamak için aradaki farkın mutlak değerini alıp tolerans değerinden küçük veya eşit olup olmadığının sorguluyoruz.

Programlama Performans İpucu: Döngüler (while – for – foreaach)



Döngüler (while – for – foreaach) içerisinde atama yapılan değişkenlerin tanımlanması döngü içerisinde değil de döngü dışında tanımlanması yapılırsa performans elde edilir. Örneğin:

```
for(int i = 0; i < 10000; i++)
{
int b = i *2;
}
```

Şeklinde b değişkeni tanımlandığında her döngüde b değişkeninin geçerlilik ömrü sona erdiğinden bellek üzerinden silinir ve b için bellek üzerinde yeni alan tahsisi yapılır, ancak bunun yerine geçerlilik ömrü bütün döngü için geçerli olacak şekilde tanımlanırsa bellek üzerinde alanın silinip yenisinin oluşturulması için gereken işlemlerden tasarruf sağlanır.

```
int b;
for(int i = 0; i < 10000; i++)
{
b = i *2;
}
```

Programlama Performans İpucu: (&&) ve (||) Operatörleri Kullanılırken

Operand (x)	operand (y)	Value of the Expression		
		x || y	x && y	x ! y
true	true	true	true	false
true	false	true	false	true
false	true	true	false	true
false	false	false	false	true

1. VE (&&) operatörünün doğru sonucu dönmesi için bütün şartlarının doğru olması gerekmektedir. Bu sebeple && operatörü kullanılırken karşılaşılan ilk yanlış şartta geriye kalan şartlar sorgulanmadan program sonraki satıra geçecektir. && operatörünün bu özelliğinden dolayı yanlış olma ihtimali yüksek olan şartlar başta yazılırsa performans artacaktır.
2. VEYA (||) operatörünün doğru sonucu dönmesi için herhangi bir şartın doğru olması yetmektedir. Bu sebeple || operatörü kullanılırken karşılaşılan ilk doğru şartta geriye kalan şartlar sorgulanmadan program sonraki satıra geçecektir. || operatörünün bu özelliğinden dolayı doğru olma ihtimali yüksek olan şartlar başta yazılırsa performans artacaktır.