C# Programlamaya Giriş

5. Hafta: Bileşik Atama ve Yineleme İfadeleri

Kodda Verimlilik ve Akış Kontrolü

Kodda Verimlilik: Neden Daha Fazla Yazasınız?

Değişkeni kendi üzerinden güncellemenin daha zarif ve kısa bir yolu vardır. İyi bir yazılımcı, kod tekrarından kaçınır.

Eski Yöntem:

```
answer = answer + 42;
```

Yeni Yöntem:

```
answer += 42;
```

- Alet Çantası: +=, -=, *=, /=, %= (Bileşik Atama Operatörleri).

- Ekstra Bilgi: += operatörü metinleri (string) birleştirmek için de kullanılabilir (source.Text += line;)

- Altın Kural: Bir değişkeni sadece 1 artırmak veya azaltmak için ++ ve -- operatörlerini tercih edin (count++;).

Döngülerin Mantığı: Tekrarı Bilgisayara Devretmek

Yineleme (Iteration), belirli bir koşul doğru olduğu sürece bir blok kodun defalarca çalıştırılmasıdır.

- **Şart Mantığı (Condition):** Döngünün devam edip etmeyeceğini belirleyen Boolean (true/false) ifade.
- **Kontrol Değişkeni (Sentinel Variable):** İterasyon sayısını veya durumunu takip eden, döngünün sonsuza girmesini engelleyen kilit değişken.

Belirsizlikte Güç: while Döngüsü

Metafor: Kapıdaki Güvenlik. Kod bloğuna girmeden önce kimlik (şart) sorar. Şart baştan yanlışsa, kod hiç çalışmaz.

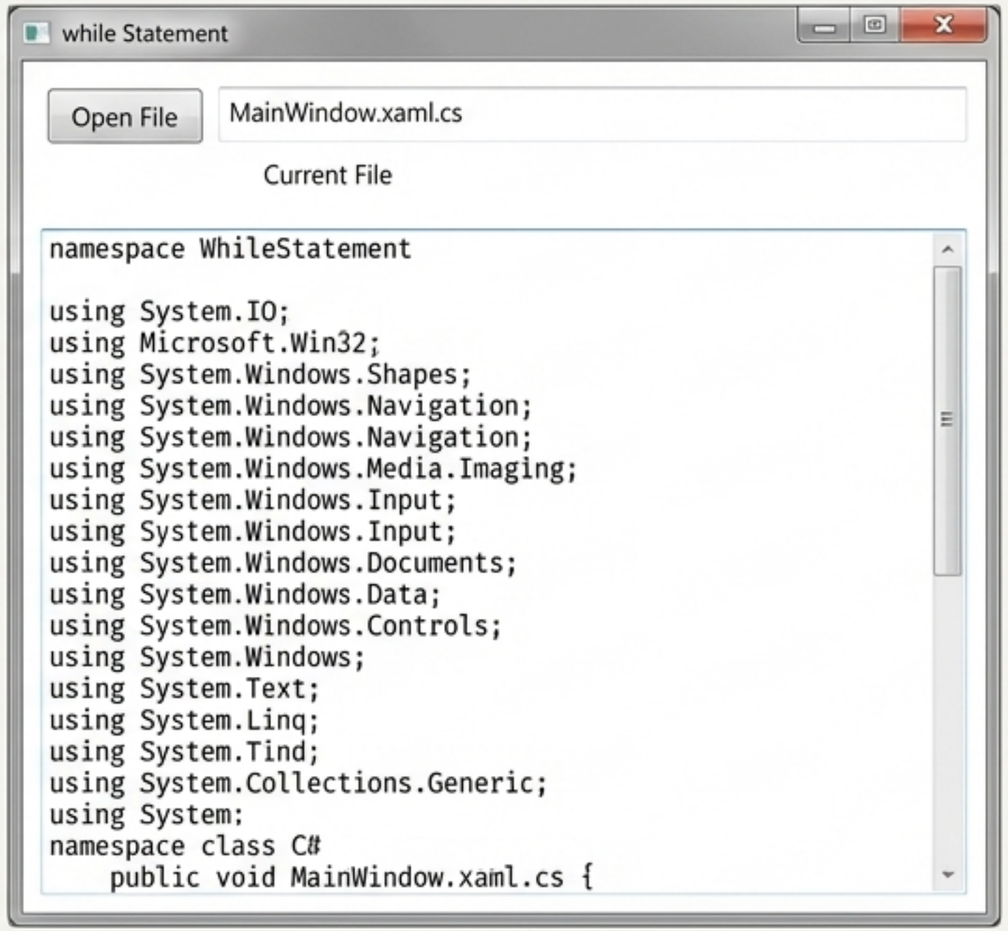
```
while (booleanExpression)
{
    // İfade 'true' olduğu sürece çalışacak kodlar
    // Kontrol değişkenini (sentinel) güncellemeyi unutmayın!
}
```

Sonsuz döngüden kaçınmak için döngü gövdesinde koşulu false yapacak bir güncelleme (örneğin i++) mutlaka bulunmalıdır.

Pratik Örnek: Dosya Okuma (while)

Bir metin belgesinde kaç satır olduğunu baştan bilemeyiz. Çözüm: Satırlar bitene kadar (null olana kadar) okumaya devam et.

```
string line = reader.ReadLine();
while (line != null)
{
   source.Text += line + '\n';
   line = reader.ReadLine(); // Kontrol
değişkenini güncelle
}
```



Belirlilikte Düzen: for Döngüsü

Metafor: Montaj Hattı. Başlangıcı, bitişi ve adımları net olan işlemler için tasarlanmıştır. while döngüsünün üç ana parçasını tek satırda birleştirir.

```
for (başlatma; koşul; güncelleme)
for (int i i = 0; i < 10; i++)
```

Başlatma (Initialization):

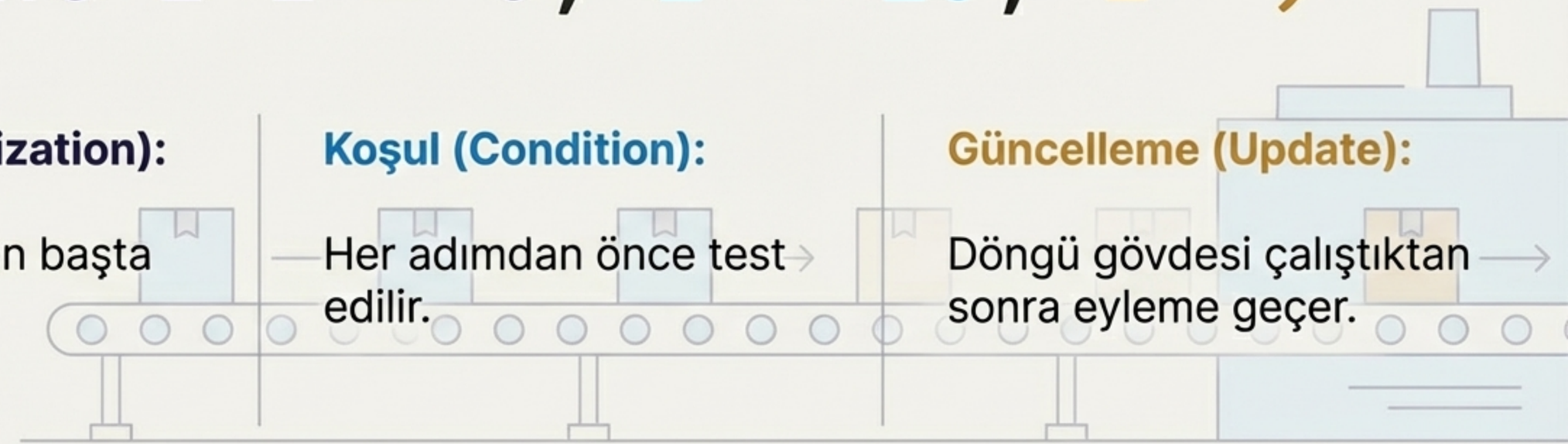
Sadece bir kez, en başta çalışır.

Koşul (Condition):

Her adımdan önce test edilir.

Güncelleme (Update):

Döngü gövdesi çalıştıktan sonra eyleme geçer.



Yaşam Döngüsü ve Kapsam (Scope)

Sınırları anlamak: for döngüsünün başlatma (initialization) kısmında tanımlanan bir değişken, döngü bittiğinde hafızadan silinir (kapsam dışına çıkar).

```
for (int i = 0; i < 10; i++)
{
    // i değişkeni sadece bu blok { } içinde yaşar
}
Console.WriteLine(i); // HATA: Derleme zamanı hatası!
```

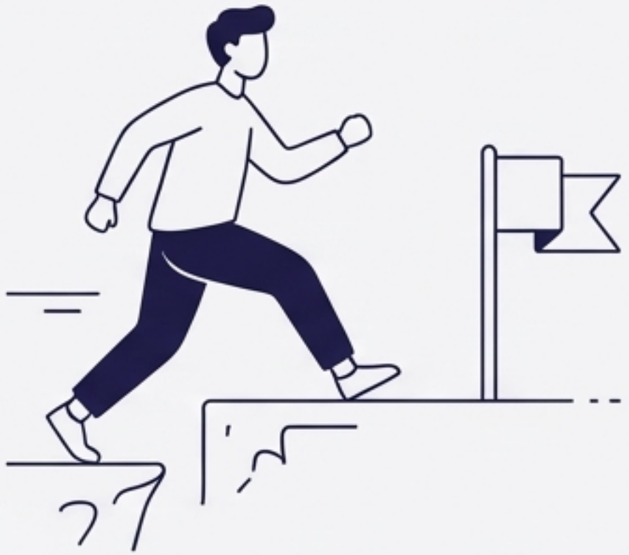
Bu kural sayesinde, arka arkaya yazdığınız farklı for döngülerinde aynı değişken adını (örneğin `i`) güvenle tekrar kullanabilirsiniz.

Önce Eylem, Sonra Kontrol: do Döngüsü

Metafor: Önce Yap, Sonra Sor. Diğer döngülerin aksine, koşulu kod bloğu çalıştıktan sonra değerlendirir.

Koşul baştan false olsa bile, döngü gövdesi **en az 1 kez** kesinlikle çalışır.

```
do
{
    // En az bir kez çalışması garanti edilen kod
}
while (booleanExpression);
```

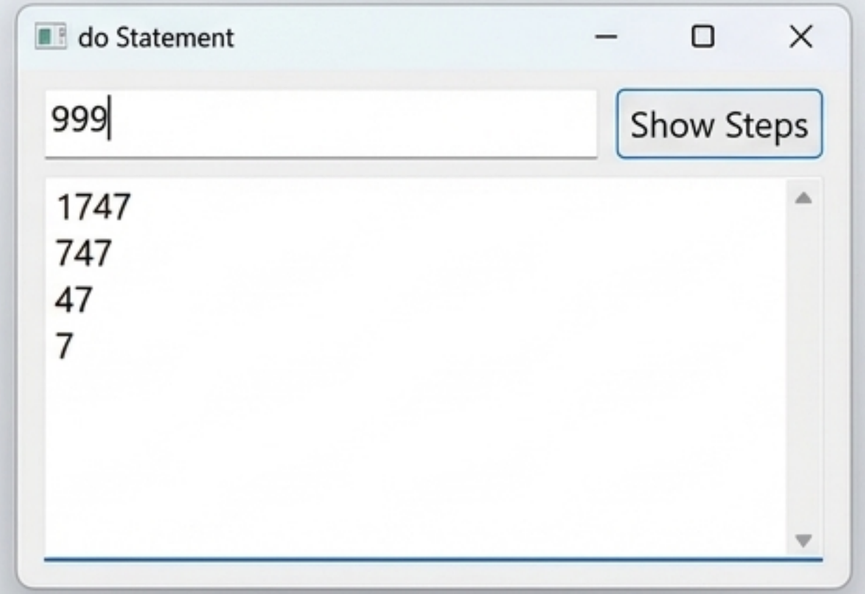


Sona eklenen noktalı virgüle dikkat!

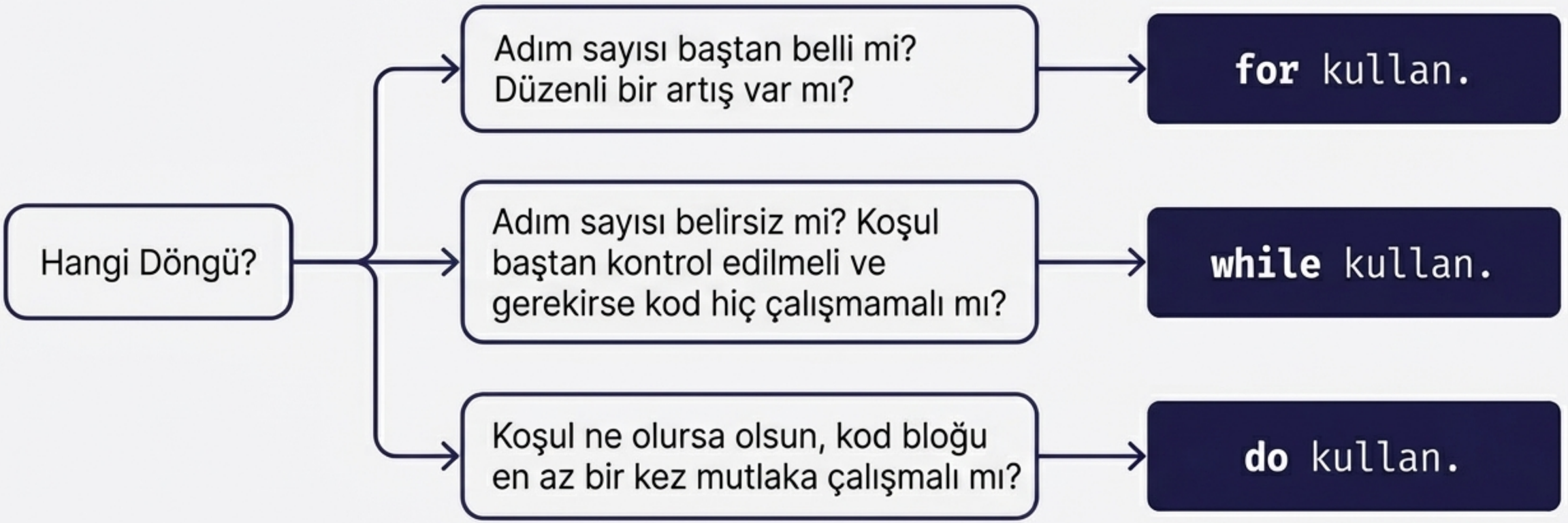
Pratik Örnek: Sayı Sistemleri Dönüşümü (do)

Neden en az bir kez çalışmaya ihtiyacımız var? Çünkü ondalık (decimal) sayı 0 bile olsa, onun sekizlik (octal) sistemde bir basamağı ("0") vardır!

```
do
{
    int nextDigit = amount % 8;
    amount /= 8; // Bileşik atama operatörü devrede!
    // ... basamakları birleştir ...
}
while (amount != 0);
```



Doğru Aracı Seçmek



İyi bir yazılımcı sadece kod yazmaz; problemi en az tekrarla, en doğru kontrol akışını (Control Flow) seçerek çözer.