Döngüler

Her fonksiyonel dilde olduğu gibi Java içerisinde de akışı düzenleyen döngüler yer almaktadır. Döngü adından da anlaşılabileceği gibi belirli veya belirsiz aralıkta tekrar sağlayan kod yapılarıdır. Java içerisinde for, while, do while ve Java 5 versiyonu ile birlikte gelen for each döngüsü ger almaktadır.

for

for döngüsü C/C++ türevi dillerde aynı özellikleri taşıyor diyebiliriz. Örnek yapısı aşağıdaki gibidir.

```
for (başlangıç ; koşul ; döngü_adımı) {
      //Çalıştırılacak kod bloğu
}
```

for içerisinde noktalı virgül ile ayrılmış 3 alan bulunuyor. Başlangıç döngü başladığı anda uğranacak adımları belirtir. Koşul döngünün her dönüşünde uğrayıp true dönmesini beklediği alandır. Döngü adımı ise eğer koşul sağlanıyorsa çalışacak alandır burada genelde sayı artırımı gibi operasyonlar geçekleştirilir.

Gerçek bir for döngüsü yazacak olursak aşağıdaki örnek verilebilir.

```
public class Ornek {
      public static void main(String[] args) {
            for (int i = 0; i < 10; i++) {
                  System.out.println("i : " + i);
            }
      }
}
```

Buradaki örnekte for içerisinde bir i değişkeni tanımlanıyor. i değeri 10 dan küçük olduğu sürecede I sayısı her dönüşte bir artırılıp döngü devam ettiriliyor. Buna göre ekrana aşağıdaki gibi bir çıktı verilir.

```
i : 0
i : 1
i : 2
i : 3
i : 4
i : 5
i : 6
i : 7
i : 8
i : 9
```

Dikkat ederseniz ekrana son olarak 10 değerini yazdırmadı. Çünkü koşulumuz i sayısının 10 dan küçük olduğu sürece dödngünün dönmesidir.

Peki döngü içerisine başlangıç koşul ve adımları yazmaksak ne olur ? Örnek olarak aşağıdaki kodun çıktısı nedir ?

```
public class Ornek {
	public static void main(String[] args) {
		for (;;) {
			System.out.println("döngü devam ediyor...");
		}
	}
}
```

Bu bir sonsuz döngüdür. Buna göre bir dışarıdan sonlandırana kadar sonsuz döngü şeklinde "devam ediyor… " yazacaktır.

Not : Sonsuz döngü bir hata olmayabilir. Belirli durumlarda sonsuz döngülere ihtiyacımız oluyor olacak.

for döngüsü içerisinde koşul ve başlangıç değerleri tek olmak zorunda değildir. Örneğin aşağıdaki gibi bir kullanımda mümkün olabilirdi.

```
public class Ornek {
	public static void main(String[] args) {
		for (int i = 0, j = 10; i < 10 && j > 0; i++, j--) {
			System.out.println("i : " + i + " j : " + j);
		}
	}
}
```

Buna göre döngü hem i hemde j değerlerini içeriyor. i değeri 10 a kadar j değeri ise 10 dan 0 a kadar devem etmektedir. Bu kodun çıktısı aşağıdaki gibidir.

```
i : 0 j : 10
i : 1 j : 9
i : 2 j : 8
i : 3 j : 7
i : 4 j : 6
i : 5 j : 5
i : 6 j : 4
i : 7 j : 3
i : 8 j : 2
i : 9 j : 1
```

while

Diğer bir döngü seçeneğimiz while döngüsüdür. Bu döngüde aşağıdaki modelde olduğu gibi sadece mantıksal koşul (true veya false) aranır.

```
while ( koşul ) {
	//Çalıştırılacak kod bloğu
}
```

Buradaki mantıksal koşul uyduğu sürece döngü devam edecektir. Aşağıdaki örnekte i sayısı tanımlanıp while içerisinde sürekli artırılıyor.

```
public class Ornek {
	public static void main(String[] args) {
		int i = 0;
		while (i < 10) {
			System.out.println("i : " + i++);
		}
	}
}
```

Dikkat : Eğer burada i değerini artırmıyor olsaydık sonsuz döngü oluşurdu.

do while

while benzeri bir başka döngüde do while döngüsüdür. Standart while' dan farkı öncelikle döngü içeriğinin çalıştırılıp sonradan koşul kontrolünün yapılmasıdır.

```
do {
	//Çalıştırılacak kod bloğu
} while ( koşul );
```

Dikkat ettiyseniz öncelikle do ile başlayan döngü içeriği çalışmaya başlanıyor sonrasında while ile koşul kontrolü yapılıyor. Buradaki önemli nokta koşul uysa da uymayasa da döngü içerisine en az bir kere girileceğidir.

Bunu bir örnek ile deneyecek olursak aşağıdaki kod çalıştırıldığı anda x değeri 2 ye eşit olmasada öncelikle döngü içeriği çalıştırıldığı için bir kere "döngü içeriği işletildi" yazacaktır.

```
public class Ornek {
	public static void main(String[] args) {
		int x = 0;
		do {
			System.out.println("döngü içeriği işletildi");
		} while ( x == 2 );
	}
}
```

for each

Java 5 versiyonu ile birlikte gelen bir döngü olan for each isim olarak böyle anılsada aslında for döngüsünün bir başka kullanım şeklidir. Amacı dizi ve liste gibi yapılar üzerinde iterasyon şeklinde gezip elemanlara ulaşmaktır. Aslında for each kullanımını diziler konusunda tekrar edecek olsakta yapısı aşağıdaki gibidir.

```
for (Tip eleman : dizi) {
	System.out.println(eleman);
}
```

Burada öncelikle elimizde bir dizi veya collection yapısında dizi isminde bir değişken olduğunu varsayıyoruz. for each bu elemanlar üzerinde dönüm ilgili tipe göre bunu eleman nesnesine atacaktır. Artık döngü içerisinde bu nesneye eleman adıyla erişebiliriz.

Örnek verecek olursa aşağıda dizinin elemanlarını ekrana yazdıran bir kod parçası bulunmaktadır.

```
public class Ornek {
	public static void main(String[] args) {
		String[] dizi = { "Ankara", "İstanbul", "İzmir" };
		for (String eleman : dizi) {
			System.out.println(eleman);
		}
	}
}
```

Bu kodun çıktısı da aşağıdaki gibi olacaktır.

```
Ankara
İstanbul
İzmir
```

İç içe döngüler

Döngü adımları kendi içlerinde de tekrar edebilir. Bu tarz döngülere iç içe döngüler diyebiliriz. Tekrar edilecek döngü sayısında bir sınır olmamakla birlikte tekrarlar kartezyen çarpım kadardır.

Örnek yapı aşağıdaki gibi olabilir.

```
for ( ... ) {
	for ( ... ) {
		for ( ... ) {
			...
		}
	}
}
```

Dikkat : Bu örnek yapıda for döngüleri içerisinde while do while gibi alternatif döngü tipleri de olabilir.

İç içe döngülere aşağıdaki gibi bir örnek verebiliriz. Bu örnek i ve j değerleri taşıyan iç içe iki döngü bulunmaktadır. Döngü değerleri çarpma şeklinde yazdırılıyor.

```
public class Ornek {
	public static void main(String[] args) {
		for (int i = 0; i < 10; i++) {
			for (int j = 0; j < 10; j++) {
				System.out.println("i * j : " + (i * j));
			}
		}
	}
}
```

Akışa Müdahale

Java içerisinde döngü akışına müdahale etmek istiyorsak break ve continue kelimelerini kullarabiliriz. Bunun haricinde return kelimesi ilgili metodtan çıkışı sağlamaktadır.

break

Döngüden tamamen çıkmaya yarar. Yani break görülen yerde döngü sonlandırılmış olur. Kullanımı aşağıdaki şekildedir.

```
for ( ... ) {
	break;
}
```

Örnek verecek olursak aşağıdaki döngü içerisinde 4 ve katı bir değer ile karşılaşılırsa döngüyü sonlandırmasını istiyoruz.

```
public class Ornek {
	public static void main(String[] args) {
		for (int i = 1; i < 10; i++) {
			if (i % 4 == 0) {
				break;
			}
			System.out.println("i : " + i);
		}
	}
}
```

Buna göre yukarıdaki kodun çıktısı aşağıdaki gibi olacaktır. Döngü 10' a kadar devam etmez çünkü 4 değerini gördüğü anda break ile sonlanacaktır.

```
i : 1
i : 2
i : 3
```

continue

break' ten farklı olarak döngünün sonlanması değil ilgili adımın pas geçilmesi isteniyorsa continue anahtar kelimesi kullanılabilir. Bu döngünün bir sonraki adımına geçmemizi sağlayacaktır. Kullanımı aşağıdaki şekildedir.

```
for ( ... ) {
	continue;
}
```

break' teki örneği aşağudaki gibi continue' da uygulayalım.

```
public class Ornek {
	public static void main(String[] args) {
		for (int i = 1; i < 10; i++) {
			if (i % 4 == 0) {
				continue;
			}
			System.out.println("i : " + i);
		}
	}
}
```

Buna göre kodumuzun çıktısı aşağıdaki gibi olacaktır.

```
i : 1
i : 2
i : 3
i : 5
i : 6
i : 7
i : 9
```

Burada break teki çıktıdan farklı olarak 4 ve katı yakalandığında döngü tamamen sonlanmayıp bir sonraki adıma geçiyor.

Dikkat : Eğer iç içe döngü kullanıyorsak break ve continue kullanımlarında aşağıdaki gibi etiketler kullanmamız gerekiyor.

```
donguBir: for ( ... ) {
	donguIki: for ( ... ) {
		donguUc: for ( ... ) {
			break donguBir;
		}
	}
}
```

return

return kelimesinin Java içerisinde iki görevi vardır. Birincisi metod sonucunda geriye değer dönmemizi sağlar diğeri ise metodtan tamamen çıkmamıza yarar. Bu konuyu metodlar konusunda işleyecek olsakta kullanımı aşağıdaki gibidir.

```
void topla(int x, int y){
	if(x < 0 )
		return;
	}
	...
}
```