**Döngü Deyimleri**

Bir programın yinelemeli olarak çalıştırılmasını sağlayan deyimlere döngü deyimleri (loop statements) denilmektedir. Java' da döngü deyimleri üçe ayrılır:

1. while döngüleri

2. for döngüleri

3. Genişletilmiş for döngüsü (enhanced for statement) (for-each döngüsü)

Genişletilmiş for döngüsü genel olarak diziler ve collection'lar ile ilgili olduğundan ileride ele alınacaktır.

**while Döngüleri**

while döngüleri kendi içerisinde ikiye ayrılmaktadır:

1. Kontrolün başta yapıldığı while döngüleri

2. Kontrolün sonda yapıldığı while döngüleri (do-while döngüleri)

**Kontrolün Başta Yapıldığı while Döngüleri**

Bir koşul sağlandığı sürece yinelemeye yol açan döngülere while döngüleri denilmektedir. Genel biçimi şöyledir:

```
while (<boolean türden ifade>)
	<deyim>
```

while döngü deyiminin parantezi içerisindeki ifade boolean türden olmak zorundadır ve boş bırakılamaz. while döngüsü şöyle çalışmaktadır: Önce parantez içerisindeki ifadenin değeri hesaplanır. Bu değer true ise döngünün içindeki deyim çalıştırılır ve başa dönülür, false ise while deyimi sonlandırılır ve akış while deyiminin sonundan devam eder. while deyiminin kendisi de dışarıdan bakıldığında tek bir deyimdir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i;

		i = 1;

		while (i < 10) {
			System.out.printf("i = %d\n", i);
			i++;
		}

		System.out.println("Program Sonu");
	}
}
```

Burada

```
while (i < 10) {

		System.out.printf("i = %d\n", i);
		i++;
	}
```

deyimi tek bir deyimdir.

Bazen while döngüleri yanlışlıkla boş deyim ile kapatılabilmektedir:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i;

		i = 1;

		System.out.println("Program Başlıyor");

		while (i < 10); //Dikkat boş deyim
		{
			System.out.printf("i = %d\n", i);
			i++;
		}

		System.out.println("Program Sonu");
	}
}
```

Bu durumda herhangi bir hata oluşmaz. Fakat şüphesiz program hatalı çalışır. Bazı durumlarda döngü deyimlerinde programcının algoritması gereği bilerek boş deyim koyması gerekebilir. Bu durumda programcı okunabilirlik açısından noktalı virgülü bir alt satıra koymalıdır:

```
while (<boolean türden ifade>)
	;
```

Program içerisinde belirli bir sayı kadar dönen döngüler çok sık olarak kullanılmaktadır. while döngü deyimi ile belirli bir sayıda dönen örnek döngü kalıpları aşağıdaki biçimlerde oluşturulabilir:

**1.** n bir tamsayı türünden değişken olmak üzere n-kez dönen döngü while döngü deyimi ile şu şekilde oluşturulabilir:

```
while (n-- > 0)
	<deyim>
```

Burada -- (azaltma) operatörünün sonek kullanımının ürettiği değer karşılaştırma işlemine sokulmaktadır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Pozitif bir sayı giriniz");
		int n = Integer.parseInt(kb.nextLine());

		while (n-- > 0)
			System.out.println("Merhaba");

		System.out.printf("n=%d\n", n);
```

```
		kb.close();
	}
}
```

Bu kalıpta dikkat edilmesi gereken n değişkeninin içerisindeki değer döngünün her adımında azaltıldığından döngüden çıkıldıktan sonra n içerisindeki değer -1 olmaktadır. Dolayısıyla eğer n değişkeni içerisindeki değer döngüden sonra gerekiyorsa döngüye girmeden önce bir değişkene atanmalı ve o değişken ile döngüye girilmelidir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Pozitif bir sayı giriniz");
		int n = Integer.parseInt(kb.nextLine());
		int temp = n;

		while (temp-- > 0)
			System.out.println("Merhaba");

		System.out.printf("n=%d\n", n);
		System.out.printf("temp=%d\n", temp);

		kb.close();
	}
}
```

**2.** n bir tamsayı türünden ve i de n ile aynı türden bir değişken olmak üzere n-kez dönen döngü while döngü deyimi ile şu şekilde oluşturulabilir:

```
i = 0;

while (i < n) {
	//...
	i++;
}
```

Burada döngü içerisinde n değişkeninin değeri herhangi bir şekilde değiştirilmediğinden döngüden sonra da kullanılabilmektedir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Pozitif bir sayı giriniz");
		int n = Integer.parseInt(kb.nextLine());

		int i = 0;

		while (i < n) {
```

```
                System.out.println("Merhaba");
                i++;
        }

        System.out.printf("n=%d\n", n);

        kb.close();
    }
}
```

**3.** n bir tamsayı ve i de n ile aynı türden bir değişken olmak üzere n-kez dönen döngü while döngü deyimi ile şu şekilde oluşturulabilir:

```
i = 0;

while (i++ < n)
    <deyim>
```

Aslında bu kalıp 2. maddede anlatılan kalıbın başka bir yazılışıdır. Bu kalıpta ++(artırma) operatörünün sonek kullanımının ürettiği değer karşılaştırma işlemine sokulmaktadır. Örneğin:

```
package csd;

class App {
    public static void main(String[] args)
    {
        java.util.Scanner kb = new java.util.Scanner(System.in);

        System.out.println("Pozitif bir sayı giriniz");
        int n = Integer.parseInt(kb.nextLine());

        int i = 0;

        while (i++ < n)
            System.out.println("Merhaba");

        System.out.printf("n=%d\n", n);

        kb.close();
    }
}
```

Bir metodun geri dönüş değeri while parantezi içerisinde bir değişkene atanacaksa operatör önceliğine dikkat edilmelidir. Atama operatörüne öncelik verilmesi için atama ifadesi parantez içerisine alınmalıdır.

Örneğin:

```
package csd;

class App {
    public static void main(String[] args)
    {
        java.util.Scanner kb = new java.util.Scanner(System.in);
        System.out.println("Sayıları girmeye başlayınız");

        int val;

        while ((val = Integer.parseInt(kb.nextLine())) != 0)
            System.out.println(val * val);
```

```
		kb.close();
	}
}
```

while ((val = Integer.parseInt(kb.nextLine())) != 0) ile önce val değişkenine değer atanacak sonrasında 0(sıfır) değeriyle karşılaştırma işlemine sokulacaktır.

while döngü deyimiyle sonsuz döngü şu şekilde oluşturulabilir:

```
while (true)
	<deyim>
```

Parantez içerisindeki ifade true değerinde olduğundan koşul her zaman sağlanacaktır.

Dikkat aşağıdaki program aslında sonsuz döngü değildir:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i;
		int n = 10;

		i = 0;

		while (i-- < n)
			;

		System.out.printf("i = %d\n", i);
	}
}
```

Burada i değeri int türünün negatif taraftan sınırları dışına çıktığında i değişkenin içerisinde int türü için pozitif en büyük değer olacağından döngü sonlanacaktır.

**Kontrolün Sonda Yapıldığı while Döngü Deyimi (do-while Döngüleri)**

Bu döngülerin çalışma biçimi kontrolün başta yapıldığı while döngüleri aynıdır. Yani while parantezi içerisindeki ifade true olduğu sürece döngü yinelenmektedir. Bu döngüler de tek fark kontrol noktasının sonra olmasıdır. Genel biçimi şöyledir:

```
do
	<deyim>
while (<boolean türden ifade>);
```

Burada koşul gerçekleşse de gerçekleşmese de akış do-while döngü deyimine geldiğinde döngü içerisindeki deyim bir kez çalıştırılır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i, n;
```

```
		i = 10;
		n = 8;

		do {
			System.out.printf("Dongu içi i=%d\n", i);
			i++;
		} while (i < n);

		System.out.printf("Dongu dışı i=%d\n", i); // i=11
	}
}
```

Burada i < n koşulu false olmasına karşın döngü içerisindeki deyim bir kez çalştırıldığıdan i değeri 11 olmaktadır.

do-while döngülerinde while parantezinin sonundaki noktalı virgül boş deyim belirtmez. Sentaksın bir parçasıdır.

do-while döngülerine seyrek gereksinim duyulmaktadır. Bazı durumlarda okunabilirliği ve algılanabilirliği artırmaktadır.

**Sınıf Çalışması:** 1(bir) den klavyeden girilen n değerine kadar tamsayıların toplamını bulan programı yazınız.

**Çözüm:**

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Pozitif bir sayı giriniz");
		int n = Integer.parseInt(kb.nextLine());

		int i;

		i = 1;

		int sum = 0;

		while (i <= n) {
			sum += i;
			i++;
		}

		System.out.printf("Toplam:%d\n", sum);

		kb.close();
	}
}
```

Yukarıdaki while döngü deyimi aşağıdaki biçimde de yazılabilir:

```
while (i <= n)
	sum += i++;
```

Burada ++ operatörünün sonek kullanımının ürettiği değer sum değişkenine eklenmektedir.

**Sınıf Çalışması:** Klavyeden sıfır girilene kadar alınan sayıların toplamını bulan programı yazınız.

**Çözüm:**

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Sayıları girmeye başlayınız");

		int val;
		int sum = 0;

		while ((val = Integer.parseInt(kb.nextLine())) != 0)
			sum += val;

		System.out.printf("Toplam:%d\n", sum);

		kb.close();
	}
}
```

**Sınıf Çalışması:** Klavyeden sıfır girilene kadar alınan sayıların çift ve tek olanlarının ayrı ayrı sayısını bulan programı yazınız.

**Çözüm:**

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Sayıları girmeye başlayınız");
		int val;
		int evenCount = 0, oddCount = 0;

		while ((val = Integer.parseInt(kb.nextLine())) != 0)
			if (val % 2 == 0)
				evenCount++;
			else
				oddCount++;

		if (evenCount != 0 || oddCount != 0) {
			System.out.printf("Çift sayıların sayısı:%d\n", evenCount);
			System.out.printf("Tek sayıların sayısı:%d\n", oddCount);
		}
		else
			System.out.println("Hiç bir değer girmediniz");

		kb.close();
	}
}
```

Program bayrak değişkeni kullanılarak da yapılabilir:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Sayıları girmeye başlayınız");

		int val;
		int evenCount = 0, oddCount = 0;
		boolean flag = false;

		while ((val = Integer.parseInt(kb.nextLine()))!= 0) {
			if (!flag)
				flag = true;

			if (val % 2 == 0)
				evenCount++;
			else
				oddCount++;
		}

		if (flag) {
			System.out.printf("Çift sayıların sayısı:%d\n", evenCount);
			System.out.printf("Tek sayıların sayısı:%d\n", oddCount);
		}
		else
			System.out.println("Hiç bir değer girmediniz");

		kb.close();
	}
}
```

**Sınıf Çalışması:** Klavyeden girilen bir sayının basamak sayısını bulan programı yazınız.

**Çözüm:**

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Bir sayı giriniz");
		int val = Integer.parseInt(kb.nextLine());
		int digitsCount = 0;
		int temp = val;

		if (val != 0)
			while (temp != 0) {
				digitsCount++;
				temp /= 10;
			}
		else
			digitsCount++;

		System.out.printf("%d sayısının basamak sayısı:%d\n", val, digitsCount);
		kb.close();
```

```
	}
}
```

Program do-while döngü deyimi kullanılarak şu şekilde yapılabilir:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Bir sayı giriniz");
		int val = Integer.parseInt(kb.nextLine());
		int digitsCount = 0;
		int temp = val;

		do {
			digitsCount++;
			temp /= 10;
		} while (temp != 0);

		System.out.printf("%d sayısının basamak sayısı:%d\n", val, digitsCount);
		kb.close();
	}
}
```

Program Utils isimli bir sınıf içerisinde getDigitsCount metodunu da yazarak şu şekilde yapılabilir:

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Bir sayı giriniz");
		int val = Integer.parseInt(kb.nextLine());

		System.out.printf("%d sayısının basamak sayısı:%d\n", val, Utils.getDigitsCount(val));
		kb.close();
	}
}

class Utils {
	public static int getDigitsCount(int val)
	{
		int digitsCount = 0;

		do {
			digitsCount++;
			val /= 10;
		} while (val != 0);

		return digitsCount;
	}
}
```

**Sınıf Çalışması:** Parametresi ile aldığı int türden bir sayının tersini döndüren getReverse metodu ve test kodunu yazınız.

**Çözüm:**

```
package csd;

class App {
	public static void main(String[] args)
	{
		java.util.Scanner kb = new java.util.Scanner(System.in);

		System.out.println("Bir sayı giriniz");
		int val = Integer.parseInt(kb.nextLine());

		System.out.printf("%d sayısının tersi:%d\n", val, Utils.getReverse(val));
		kb.close();
	}
}

class Utils {
	public static int getReverse(int val)
	{
		int result;

		result = 0;

		while (val != 0) {
			result = result * 10 + val % 10;
			val /= 10;
		}

		return result;
	}

	public static int getDigitsSum(int val)
	{
		int sum = 0;

		do {
			sum += val % 10;
			val /= 10;
		} while (val != 0);

		return Math.abs(sum);
	}

	public static int getDigitsCount(int val)
	{
		int digitsCount = 0;

		do {
			digitsCount++;
			val /= 10;
		} while (val != 0);

		return digitsCount;
	}
}
```

**Sınıf Çalışması:** Üç basamaklı Armstrong sayılarını bulup ekrana yazdıran programı yazınız. NOT:Üç basamaklı Armstrong sayıları, basamaklarının küpleri toplamı sayınının kendisine eşit olan sayılardır.

**Çözüm:**

```
package csd;
```

```
class App {
	public static void main(String[] args)
	{
		int i;

		i = 100;

		while (i < 1000) {
			int temp = i;
			int cubeSum = 0;

			do {
				int digit = temp % 10;

				cubeSum += digit * digit * digit;
				temp /= 10;
			} while (temp != 0);

			if (i == cubeSum)
				System.out.println(i);

			i++;
		}
	}
}
```