Metodlar

Metodlar Java içerisinde iş yapan ve fonksiyonalite sağlayan tanımlardır diyebiliriz. Bu başka dillerde fonksiyon veya prosedür olarak ta adlandırılabilir. Java dilinde metod kavramı prosedür ve fonksiyondan farklı olarak dönüşlü ve dönüşsüz olmak üzere ikiye ayrılır.

Metod tanımı aşağıdaki gibidir.

```
[void veya dönüş_tipi] [metod_adı] (parametre1, parametre2){
      metod içeriği
}
```

Buna göre dönüşlü metoda aşağıdaki örnek verilebilir.

```
int topla(int x, int y){
      int sonuc = x + y;
      return sonuc;
}
```

Bu örnekte topla isminde bir metodumuz var ve geriye int tipinde bir değer dönüyor. Bunun yanında x ve y adında iki adet parametre alıyor. Metodumuzun başında int yazdığı için sadece int tipinde bir değer dönebilir. Aksi durumda derleme anında hata verecektir. Metodun değer dönüşünü sağlayan kelime return' dür. Daha onceki konularda da return kelimesinin bir başka işinin de metodlardan çıkmak olduğundan bahsetmiştik.

Peki bu metodu nasıl kullanabiliriz ? Aşağıdaki örnekte topla metodunu main içerisinde kullanıyoruz.

```
public class Ornek {

      int topla(int x, int y) {
            return x + y;
      }

      public static void main(String[] args) {
            Ornek orn = new Ornek();
            int toplam = orn.topla(4, 6);
            System.out.println("toplam : " + toplam);
      }
}
```

Dikkat : Metoda erişmek için nesneyi oluşturmamız gerekti. Bunun nedeni metoda erişmek için referansa ihtiyaç duymamızdı.

Dönüşsüz metodlar geriye bir değer döndürmeyen sadece kendi içinde işlem yapan metodlardır. Bunlar prosedür gibide düşünebiliriz.

Dönüşsüz metod kullanımı aşağıdaki gibidir.

```
void bilgiYazdir(String ad, String soyad) {
	System.out.println(ad + " " + soyad);
}
```

Buradaki örnekte metodun void ile başlaması geriye bir değer döndürmeyeceğini ifadece eder. Bunu bir sınıf içerisinde örneklendirecek olursak aşağıdaki gibi uygulama yazabiliriz.

```
public class Ornek {

	void bilgiYazdir(String ad, String soyad) {
		System.out.println(ad + " " + soyad);
	}

	public static void main(String[] args) {
		Ornek orn = new Ornek();
		orn.bilgiYazdir("Melih", "Sakarya");
	}
}
```

Metodların belirli özellikleri vardır. Bunları daha net görmek açısından aşağıdaki gibi sıralabiliriz.

- İstenilen tip ve sayıda parametre alabilirler.
- return kelimesini gördükleri yerde metod sonlandırılmış olur.
- void metodlar geriye değer döndürmezler.
- Dönüşlü metodlar geriye dönüş tipinde değer döndürmek zorundadırlar.
- Metod alanı süslü parantez ile belirtilmek zorundadır.
- Nesne parametresi alabilirler.
- Geriye nesne parametresi döndürebilirler.

Dikkat : Daha önce de belirttiğimiz gibi metod isimlendirilmesi küçük harf ile başlar ve takip eden kelimeler büyük harf ile başlatılır. Bu bir isimlendirme standartıdır ve uyulmaması derleme hatasına neden olmaz. Buna "toplamaIslemi" , "bilgiYazdirmaMetodu" gibi örnekler verilebilir.

Nesne ve metod değişkenleri

Sınıflar içerisinde kullandığımız değişkenler ile metodlar içerisinde kullandığımız değişkenler arasında davranışsal olarak farklar vardır. Aşağıdaki örnekte sınıf ve metod değişkenleri bulunmaktadır.

```
public class Ornek {

	int x; //Nense değişkeni

	public static void main(String[] args) {

		int y; //Metod değişkeni

	}
}
```

Buradaki x nesne değişkenini, y ise metod değişkenini ifade etmektedir. Peki bunlara erişmeye çalışacak olursak ne yapmalıyız ? Birincisi nesne değişkenine main metodundan erişeceksek bu durumda Ornek sınıfından bir nesne yaratıp x değişkenine bu şekilde erişmeliyiz. y değişkeni ise yerel bir değişken olduğundan sadece metod içerisinden erişebilirdir.

Metod Değişkenleri	Nesne Değişkenleri
Değişkenleri kullanmak için ilk değer atamaları yapılmak zorundadır.	İlk değerler atanmazsa ilkel tipler için default değerler, nesne tipleri için null referans alırlar.
Sadece metod içerisinden erişilebilirler.	Nesne oldukça ve izinler dahilinde dışarıdan erişebilirler.

Örnek vermek gerekirse aşağıdaki kod içerisinde y değerini yazdırken hata alacağımızdan bahsediliyor. Bunun nedeni y değişkeninin bir metod değişkeni yani lokal değişken olması ve ilk değerinin atanmamasıdır. Eğer x değişkenini yazdıracak olursak int' in ilkel tip olmasından dolayı 0 değerini verecektir.

```
public class Ornek {

	int x; //Nense değişkeni

	void deneme(){
		int y; //Metod değişkeni
		System.out.println(x);
		System.out.println(y); //hata verir...
	}

	public static void main(String[] args) {

		Ornek orn = new Ornek();

		orn.deneme();
	}
}
```

Metod çağırımları

Metod çağırımlarında belirli kurallar vardır. Eğer bir metod dışarıdan çağırılacaksa sınıfın nesnesinin yaratılması gerekir ancak aynı sınıf üzerindeyse direkt olarak erişelebilirler.

Dikkat : Bu kural static metodlar için geçerli değildir ve static herhangi bir metod nesne metoduna direkt olarak erişemez.

Örnek vermek gerekirse aşağıdaki metodlar birbirlerini direkt olarak çağıdabiliyorlar ancak main metodu static olduğu için yukarıdaki metodları çağırmak için sınıfın bir nesnesini yaratmak zorunda kalıyor.

```
public class Ornek {
```

```
	void topla(int x, int y) {
		System.out.println(x + y);
	}

	void cikar(int x, int y) {
		System.out.println(x - y);
	}

	public static void main(String[] args) {

		Ornek orn = new Ornek();

		orn.topla(2, 6);
		orn.cikar(8, 4);
	}
}
```

Yukarıdaki örnekteki topla metodu dışarıdaki bir sınıftan çağırılacak olsaydı yine Ornek sınıfından bir nesne yaratmamız gerekecekti.

Metod overloading (aşırı yükleme)

Bir sınıf içerisinde aynı isimde birden fazla metod olabilir mi ? Evet olabilir ve buna metod overloading yani aşırı yükleme denir. Örnek olarak aşağıdaki sınıfı inceleyelim.

```
public class Matematik {

	int topla(int x, int y) {
		return x + y;
	}

	public static void main(String[] args) {
		Matematik mat = new Matematik();
		int sonuc = mat.topla(2, 6);
	}
}
```

Bu sınıfta topla isminde bir metod var ve içerisinde iki tane int tipinde parametre alıyor. Peki ben bu metoda üç veya dört adet parametre göndersem ? Başka bir seçenekte bir float birde double göndermek isteyebilirim de ? Bu durumda her seferinde yeni metod ismi yazmak yerine metod overloading özelliğini kullanabilirim. Aşağıdaki örnekte matematik sınıfı içerisinde birden fazla topla metodu bulunuyor.

```
public class Matematik {

	int topla(int x, int y) {
		return x + y;
	}

	int topla(int x, int y, int z) {
		return x + y + z;
	}

	float topla(int x, float y) {
```

```
            return x + y;
      }

      public static void main(String[] args) {
            Matematik mat = new Matematik();
            int sonuc1 = mat.topla(2, 6);
            int sonuc2 = mat.topla(2, 6, 8);
            float sonuc3 = mat.topla(2, 3.14f);
      }
}
```

Metod overloading kullanabilmek için belirli kurallar vardır. Örneğin dönüş tipleri aynı isimli metodları ayrıştırmak için bir seçenek değildir. Bu kuralları sayacak olursak.

- Parametre tipleri farklı olmalı
- Parametre sayıları farklı olmalı

Bunun haricindeki kurallar metod overloading için yeterli değildir.

main metodu

Baştan beri yazdığımız tüm sınıflarda operasyonel işlemlerimizi main isminde bir metod içerisine yazdık. Peki bu metodun özelliği nedir ? Main metodunu en baştaki açıklamalarımızda ana thread mekanizması olarak ilk uğranan metod olarak tanımlamıştık. Bunun bir anlamıda Java içerisinde biz her ne kadar satır satır yazsakta aslında çalıştırılabilir sınıflarda ilk uğranan metodun main olduğudur.

```
public class Ornek {

      public static void main(String[] args) {

      }
}
```

Peki main metodu içerisinde yer alan String dizisi şeklinde gelen args parametresi ne işe yarıyor ? Bu parametre çalıştırılabilir sınıfların ilk argumanlarını almasını sağlayan bir değişken. Örnek vermek gerekirse aşağıdaki kodu derledikten sonra bir sonraki satırdaki gibi parametre gönderebiliriz.

```
public class Ornek {

      public static void main(String[] args) {
            System.out.println("parametre-1 : " + args[0]);
            System.out.println("parametre-2 : " + args[1]);
      }
}
```

java Ornek "1. parametre" "2. parametre"

varargs

Aşağıda yazdığımız sınıf içerisinde üç adet topla metodu var. Bunun amacı iki, üç veya 4 sayıyı toplayabilecek bir metod yapmaktı. Peki dört, beş veya yüz adet parametre gönderebileceğim sınırsız sayıda değişken alabilecek bir metodum olsun istesem ?

```
public class Ornek {

	void topla(int x, int y){
		System.out.println(x + y);
	}
	void topla(int x, int y, int z){
		System.out.println(x + y + z);
	}
	void topla(int x, int y, int z, int t){
		System.out.println(x + y + z + t);
	}

	public static void main(String[] args) {

		Ornek orn = new Ornek();
		orn.topla(2, 4);
		orn.topla(2, 4, 6);
		orn.topla(2, 4, 6, 8);
	}
}
```

Burada sınırsız parametre alımı için Java SE 5 versiyonu ile gelen varargs yani değişken uzunlukta arguman tanımlarını kullanabiliriz. Varargs kullanımında sınırsız parametre aşağıda olduğu gibi ... yani üç nokta ile belirtiliyor.

```
public class Ornek {

	void topla(int ... x){
		int toplam = 0;
		for (int deger : x) {
			toplam += deger;
		}
		System.out.println(toplam);
	}

	public static void main(String[] args) {

		Ornek orn = new Ornek();
		orn.topla(2, 4);
		orn.topla(2, 4, 6);
		orn.topla(2, 4, 6, 8);
	}
}
```

Dikkat ettiyseniz az önce üç metod ile yapabildigim işlemi şimdi tek metod ile yapabildim. Varargs ile tanımlanmış parametreler metod içerisine dizi olarak verilmektedir. Bu yüzden toplam metodu içerisinde gelen x parametresini for each döngüsü ile gezip toplam değişkenine ekliyorum.

Peki birden fazla değişken parametresinin geldiği bir uygulamada varargs kullanımı çakışma yapmaz mı ? Aslında yapabilir ancak bunu engellemek için aşağıdaki gibi kurallar eklenmiştir.

- Bir metod içerisinde sadece bir adet varargs parametresi olabilir.
- Birden fazla parametre geliyorsa son parametre varargs olmalıdır.

Bunu örnekleyecek olursak aşağıdaki gibi bir kullanım hatalıdır.

```
void topla(int ... x, double ... y){
}

void topla(int ... x, double y){
}
```

Aşağıdaki kullanımlar ise doğrudur.

```
void topla(int x, double ... y){
}

void topla(int x, boolean durum , double y){
}
```

Yinelemeli (Recursive) metodlar

Recursion kavramı kendini yinelemekten gelmektedir. Programlama dilleri içerisinde de tekrar tekrar çağırılan metodlar recursive metodlar olarak adlandırılır. Bir örnek vermek gerekirse faktoriyel hesabını yapan bir kodu nasıl yazarsınız ?

```
public class Ornek {

	int faktoriyel(int x){
		if(x>0){
			return x * faktoriyel(x-1);
		}
		else{
			return 1;
		}
	}
	public static void main(String[] args) {

		Ornek orn = new Ornek();
		int f1 = orn.faktoriyel(4);
		System.out.println("sonuc : " + f1);

	}
}
```

Yukarıdaki örnekte faktoriyel hesabı yapan recursive bir fonksiyon bulunmaktadır. Buna göre her hesaplama sonrasında sayı 0’ dan büyük olduğu sürece tekrar kendini çağırmakta ve 0’ a kadar devam etmektedir.

Yukarıdaki kodun çıktısı 4’ ün faktoriyeli yani 24’ tür.