Semih Kırdinli

Java Developer at Solvia

Linkedin : https://www.linkedin.com/in/semihkirdinli/
Github : https://github.com/semih

Fonksiyonlar

- Metotlara veya Fonksiyonlara Neden İhtiyaç Duyarız?
- Metotları Nasıl Tanımlarız?

Dizi Tanımı ve Operasyonları

Matris Tanımı ve Operasyonları

Örnekler

Fonksiyonlar

Metotlar veya diğer adıyla **fonksiyonlar** belli bir işi yapan ve sürekli kullanılabilen yapılardır. **Metotlar** veya **fonksiyonlar** bir kez tanımlanır. Her çağrıldığında fonksiyon bloğundaki işler yapılır.

Java'da genelde **fonksiyonlara** *metot* deriz. Diğer bazı programlama dillerindeki gibi fonksiyon demeyiz. Çünkü metotlar class'lara özgüdür. Bu yüzden Java programlama dilinde fonksiyonlara metot demek daha doğru.

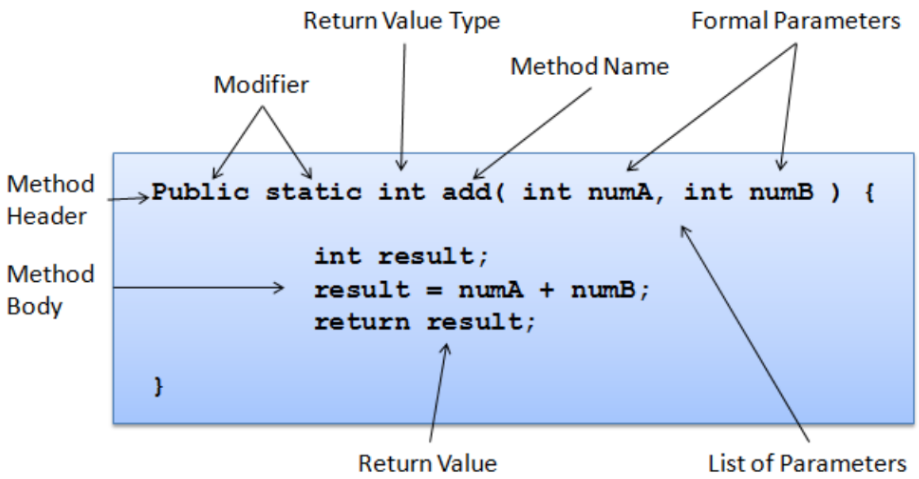
Ekrana bir şey yazdırmak için kullandığımız **System.out.println** da aslında bir metottur. System paketinin içindeki **out** un içindeki **println** fonksiyonu. Ve biz bu **fonksiyona** bir değer gönderiyoruz o da ekrana yazdırıyor. Aslında tek işlemi ekrana yazı yazdırmak.

Metotlara veya Fonksiyonlara Neden İhtiyaç Duyarız?

Metotlar olmadan da kod yazılabilir. Ancak çok büyük Java projelerinde bir işlemi 10 15 defa yapmamız gerekebilir. İşte böyle durumlarda yaptığımız işlemi bir **metot** olarak yazarsak ihtiyaç duyduğumuz yerde bu **metodu** çağırabiliriz. Bir defa yazarak her yerde kullanabiliriz.

Metotları Nasıl Tanımlarız?

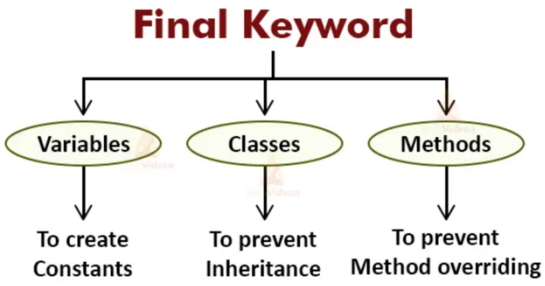
Metotları tanımlarken aslında birçok anahtar kelime kullanırız, şöyle ki;



Şimdi metot tanımlamalarındaki alanların ne işe yaradığını öğrenelim.

Access Modifier: Erişim belirteci olarak kullanılan alandır. Metoda nasıl erişilmesi gerektiğini belirtir. (public, protected, private, static, final) (Bunların ne olduğu derste ayrıntılı bir şekilde anlatılacaktır.)

public	Diğer class'lardan ve paketlerden erişilebilir.
protected	Aynı paket veya alt sınıflardan erişilebilir.
<default, no access modifier>	Sadece aynı paket içindeki sınıflardan erişilebilir.
private	Sadece aynı class içinden erişilebilir.
static	static anahtar kelimesi kullanılarak oluşturulan değişkenler nesne değişkeni değil "sınıf değişkeni" olarak adlandırılırlar. Sınıf değişkenleri içinde tanımlandığı sınıftan hiçbir nesne oluşturulmamış olsa bile bellekte yer kaplar. Nesne değişkenleri ise ancak bir nesne tanımlandığında bellekte yer kaplar. Bu iki tür değişkenin ayrıldığı diğer nokta ise sınıf değişkeninin sadece bir örneğinin olmasıdır. Yani o sınıftan kaç tane nesne oluşursa oluşsun, bellekte tek bir tane sınıf değişkeni vardır ve her zaman aynı sınıf değişkenine erişilir. Değişkenlerde olduğu gibi metotlarda da static anahtar kelimesi kullanılarak nesnelerden bağımsız sınıf metotları yazılabilir. Sınıf metotlarını çağırmak için metodun bulunduğu sınıftan bir nesne oluşturmaya ihtiyaç yoktur. Değişkenlere erişir gibi direk sınıf ismi ile bu metotları çağırmak mümkündür. static bir metot içinden static olmayan değişkenlere erişilemez ve static olmayan metotlar çağrılamaz.
final	Bir metot final olarak tanımlanmışsa, metodun tanımlı olduğu class'tan kalıtım alındığında, ilgili metot override edilemez.



Return Value Type: Dönüş tipi. Metot bir değer dönecekse, bu değerin veri yapısını belirtir. Metot karakter içeren bir veri tipi dönecekse String kullanılabilir. Metodun herhangi bir değer dönmesi istenmiyorsa **void** anahtar kelimesi kullanılmalıdır.

Method Name: Metoda verilecek ismi belirtir. Java'da metotlar küçük işlemler yapan bloklardır. Bu nedenle metot ismi seçerken, metodun içinde yapılan işlemi anlatması çok önemlidir. Çünkü anlamsız olarak isimlendirilmiş metotların ne yaptığını anlamak için kodu incelemek durumunda kalabilirsiniz.

Parametre Listesi: Metodun alacağı parametreler. Herhangi bir tipte olabilir. Metot parametrelerine access modifier yazılamaz. Metodun dışından erişilemez.

Pass by value vs. Pass by reference

Java programlama dilinde metotlara parametreler Pass by value olarak aktarılır. Yani, primitive tipte bir veri tipi metoda parametre olarak gönderilirse, bu veri tipinin bir kopyası metoda parametre olarak JVM tarafından gönderilir. Gönderilen parametrede herhangi bir değişiklik yapılsa dahi metot sonlandığında bu değişiklik görülmez. Buna Pass by Value denir.

Olayı biraz daha anlamak adına hemen aşağıdaki örneği inceleyelim.

```java
public static void main(String[] args) {
    int a = 10;
    int b = 11;
    passByValue(a, b);
    System.out.println("Value of a = " + a);
    System.out.println("Value of b = " + b);

}

public static void passByValue(int a, int b) {
    System.out.println("Value of a = " + a);
    System.out.println("Value of b = " + b);

    int newValueA = a; // Gelen a değerini yeni bir değere atıyoruz.
    int newValueB = b; // Gelen b değerini yeni bir değere atıyoruz.

    newValueA = 20; // değeri değiştiriyoruz.
    newValueB = 30; // değeri değiştiriyoruz..

    System.out.println("Value of newValueA = " + newValueA);
    System.out.println("Value of newValueB = " + newValueB);

    a = 100;
    b = 200;

    System.out.println("Value of a = " + a);
    System.out.println("Value of b = " + b);


}
```

```
Value of a = 10
Value of b = 11
Value of newValueA = 20
Value of newValueB = 30
Value of a in method = 100
Value of b in method = 200
Value of a after method executed = 10
Value of b after method executed = 11
```

Şimdide Pass By Reference olarak Java nasıl çalışıyor inceleyelim.

```java
public class PassByReference {

    public static void main(String[] args) {

        Foo a = new Foo( number: 1);
        Foo b = new Foo( number: 1);

        System.out.println("Before Modification");
        System.out.println("Value of a is " + a.number);
        System.out.println("Value of b is " + b.number);
        System.out.println();

        modify(a, b);

        System.out.println("After Modification");
        System.out.println("Value of a is " + a.number);
        System.out.println("Value of b is " + b.number);
    }

    public static void modify(Foo a1, Foo b1) {
        a1.number++;

        b1 = new Foo( number: 1);
        b1.number++;
    }
}

class Foo {
    public int number;

    public Foo(int number) { this.number = number; }
}
```

```
Before Modification
Value of a is 1
Value of b is 1

After Modification
Value of a is 2
Value of b is 1

Process finished with exit code 0
```

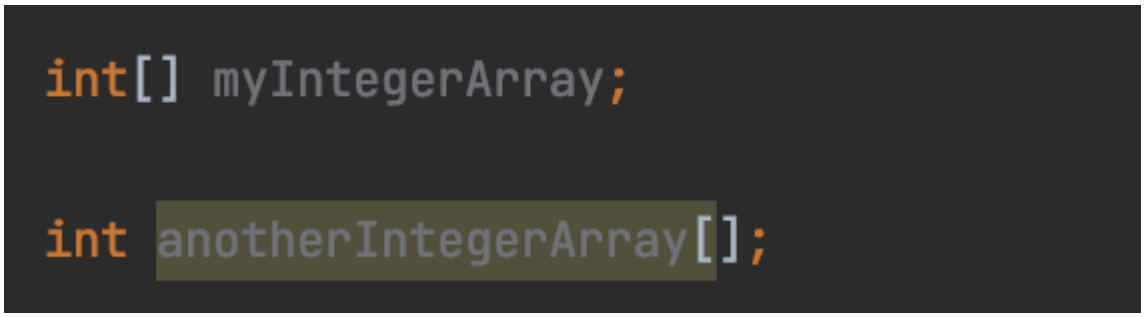
Dizi Tanımı ve Operasyonları

Dizi (Array) kavramı programlama dillerinde bir veri tipini ifade eder. Bu veri tipi liste halindeki ardışık verileri bir arada tutan yapıya denilir. Bu ardışık yapıya ait elemanlara indeks yoluyla erişim sağlanabilir.

- Diziler sabit boyutludur. (Örneğin: 10 elemanlık dizi.)
- Dizilerde aynı tipten veri tutulur. (Örneğin: tüm elemanları “int” olan bir dizi.)
- Dizi bir defa oluşturulduktan sonra boyutu değiştirilemez.

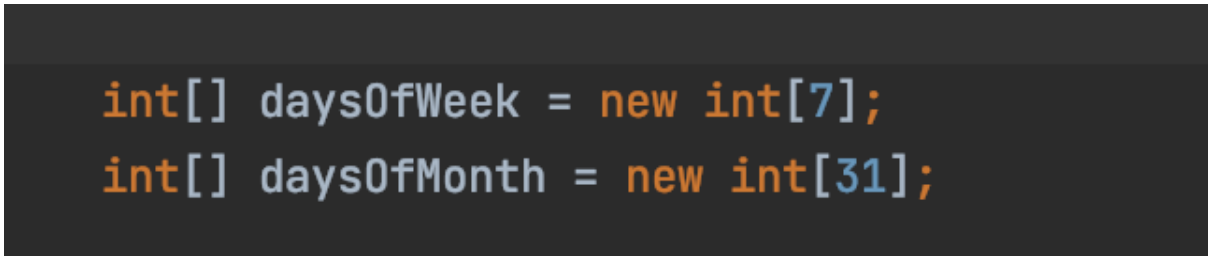
Dizi’nin hafızada bir başlangıç adresi olur ve ardışık olan diğer elemanlar sırayla hafızaya yerleştirilir. Dizi’ler “new” anahtar sözcüğüyle oluşturulur. Böylece, Heap Hafıza bölgesinde yer kaplarlar.

Java'da diziler iki farklı şekilde tanımlanabilir.



Best practice olarak ilk yöntemin tercih edildiğini söyleyebiliriz. İkinci yöntem C Style olarak adlandırılır.

Yukarıdaki dizi tanımlanmasında bellekte herhangi bir alan ayrılmamıştır. Her iki diziyi kullanabilmek için öncelikle bellekte yer ayırmak gerekiyor. İleride daha fazla bahsedeceğimiz üzere, diziler için bellekte yer ayırmak için " **new** " anahtar kelimesini kullanıyoruz.



Yukarıda iki farklı dizi tanımlaması yaptık.

Birinci dizi olan **daysOfWeek** dizisi ile haftanın günlerini rakamsal olarak tutabileceğimiz bir dizi tanımlaması yaptık.

İkinci dizi olan **daysOfMonth** dizisi ile ayın tüm günlerini rakamsal olarak tutabileceğimiz bir dizi tanımlaması daha yaptık.

Tanımlanan dizi hafızada sırası ile tutulur. "**daysOfWeek**" değişkeni dizinin başlangıç adresini tutar. Dizilerde ardışık bir sıra olduğu için ilk elemandan sonra gelen elemanların hafıza adresleri de birer birer artar. Dizi blok halinde yer kaplar. Diziye erişmek için indeks numarası kullanılır. Örneğin: "**daysOfWeek[0]**" demek dizinin 1. Elemanını verecektir. Java'da dizilerin indeksleri sıfırdan başlar. Son indeks array.length-1'dir. Örneğin: "**daysOfWeek[5]**", 5 nolu indeksteki dizi elemanını ver dediğimizde aslında dizinin 6. elemanına erişmiş oluruz. Valid olmayan bir indeks değerine erişilmeye çalışıldığında **ArrayIndexOutOfBoundsException** hatası alınır.

Dizilerin ne olduğunu anladığımıza göre, şimdi diziler üzerinde ne tür işlemler yapabiliyoruz örneklerimiz ile inceleyelim.

```java
String[] daysOfWeek = new String[7];
daysOfWeek[0] = "PAZARTESİ";
daysOfWeek[1] = "SALI";
daysOfWeek[2] = "ÇARŞAMBA";
daysOfWeek[3] = "PERŞEMBE";
daysOfWeek[4] = "CUMA";
daysOfWeek[5] = "CUMARTESİ";
daysOfWeek[6] = "PAZAR";

// Tanımlama, oluşturma ve ilk değer atamanın aynı anda yapılması.
String[] preDefinedDaysOfWeek = {"PAZARTESİ", "SALI", "ÇARŞAMBA",
"PERŞEMBE", "CUMA", "CUMARTESİ", "PAZAR"};
```

Yukarıdaki örnekte dizilerin tanımlamasına daha farklı bir örnek verdik. Şimdi bu diziler üzerinde nasıl işlem yapabileceğimize bakalım.

```java
// Dizinin içindeki tüm elemanları yazdıralım.
// Bunun için daha önce de öğrendiğimiz for döngüsünü kullanacağız.
// Gördüğünüz gibi öncelikle for döngüsünün başlayacağı index numarasını yazıyoruz.
// index numarası dizinin boyutundan küçük olduğu sürece işleme devam ediyoruz.
// son olarakta index numarasını arttırıyoruz.
for (int index = 0; index < preDefinedDaysOfWeek.length; index++) {
    System.out.format("Haftanın %d inci günü = %s %n", index + 1, preDefinedDaysOfWeek[index]);
}

System.out.println("********************");


// Foreach döngüsü, diziler ve Collection arayüzünden miras almış sınıflar içerisinde iterasyon yapmak için kullanılır.
// Bu döngü mekanizmasında index numarasına ihtiyaç duymuyoruz, çünkü iterator ile dönmeye başlıyoruz.
// Iterator konusu Collection konusu olduğu için ayrıca araştırmanızı tavsiye ederim :)
int dayNumber = 1;
for (String day : daysOfWeek) {
    System.out.format("Haftanın %d inci günü = %s %n", dayNumber++, day);
}
//Gördüğünüz gibi elimizde index numarası olmadığından dolayı, dışarıdan tanımladığımız bir counter
// aracılığı ile haftanın günlerinizi yazdırdık.
```

```
Haftanın 1 inci günü = PAZARTESI
Haftanın 2 inci günü = SALI
Haftanın 3 inci günü = ÇARŞAMBA
Haftanın 4 inci günü = PERŞEMBE
Haftanın 5 inci günü = CUMA
Haftanın 6 inci günü = CUMARTESI
Haftanın 7 inci günü = PAZAR
********************
Haftanın 1 inci günü = PAZARTESI
Haftanın 2 inci günü = SALI
Haftanın 3 inci günü = ÇARŞAMBA
Haftanın 4 inci günü = PERŞEMBE
Haftanın 5 inci günü = CUMA
Haftanın 6 inci günü = CUMARTESI
Haftanın 7 inci günü = PAZAR
```

Şimdi bir dizinin fonksiyonlara parametre olarak nasıl gönderildiğini görelim.

```java
public static void throwExceptionIfEmptyArray(String[] stringArray) {
    if (stringArray.length == 0) {
        throw new IllegalStateException("Array is empty");
    }
}

public static boolean hasGivenValueInArray(String[] array, String value) {
    throwExceptionIfEmptyArray(array);
    for (int index = 0; index < array.length; index++) {
        if (value.equals(array[index])) {
            return true;
        }
    }

    return false;
}
```

Şimdi de bir dizinin nasıl method parametresi olarak döndürüldüğünü görelim.

```java
public static char[] getCharacters(String givenValue) {
    // Null ve empty kontrolümü
    if (givenValue == null || givenValue.isEmpty()) {
        return new char[]{};
    }
    return givenValue.toCharArray();
    // String sınıfına ait charArray methodu ile tüm karakterleri bir dizi olarak döndürdük..

}
```

Dizi İşlemleriyle İlgili Bazı Metotlar:

```java
char[] a1 = {'s','o','l','v','i','a'};
char[] a2 = {'a','b','c','d','e','f'};

System.arraycopy(a1, 2, a2, 3, 3);

char[] b1 = {'s','o','l','v','i','a'};
char[] b2 = Arrays.copyOf(b1, 7);

Arrays.sort(b1);
boolean isTheSame = Arrays.equals(b1, b2);

String arrayAsString = Arrays.toString(daysOfWeek);
```

Matris Tanımı ve Operasyonları

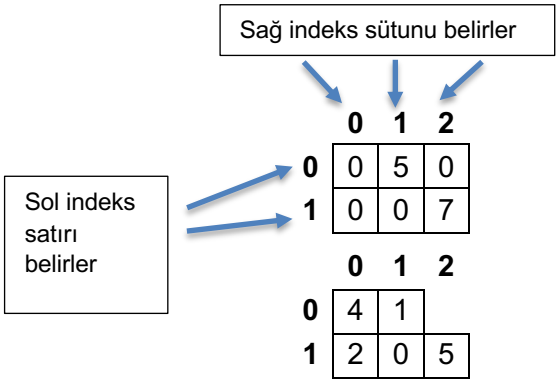
Java'da diziler birden fazla boyuta sahip olabilir. Çok boyutlu diziler, aslında dizilerden oluşan dizilerdir. Çok boyutlu dizi değişkeni tanımlamak için her bir indeks, ek bir küme parantezi kullanılarak belirtilir.

Örneğin aşağıdaki ifade **twoD** adında, iki boyutlu bir dizi değişkeni tanımıdır.

```
int[][] twoD = new int[4][5];
```

```
int[][] matrix = new int[2][3];
matrix[0][1] = 5;
matrix[1][2] = 7;

int[][] matrix2 = {{4,1},{2,0,5}};
```



Çok boyutlu bir dizi için bellekte yer ayırırken, yalnızca ilk boyut (en soldaki) için bellek büyüklüğünü tanımlamak yeterlidir. Kalan boyutlar için bellekte ayrıca yer ayırabilirsiniz. İkinci boyut için yer ayırma sonradan manuel olarak yapılır.

```
int[][] twoD = new int[4][];
twoD[0] = new int[5];
twoD[1] = new int[5];
twoD[2] = new int[5];
twoD[3] = new int[5];
```

Aşağıdaki örnekte foreach ve for döngüleri ile bir matristeki elemanların değerlerine erişim yapılabilir.

```
char[][] matrix = {
        {'A', 'B', 'C', 'D', 'E'},
        {'F', 'G', 'H', 'I', 'K'},
        {'L', 'M', 'N', 'O', 'P'},
        {'Q', 'R', 'S', 'T', 'U'},
        {'V', 'W', 'X', 'Y', 'Z'},
};
```

```
for (char[] row : matrix) {
    for (char value : row) {
        System.out.format("%c ", value);
    }
    System.out.println();
}

System.out.println("********************");

for (int row=0;row<matrix.length;row++) {
    for (int col=0;col<matrix[row].length;col++){
        System.out.format("%c ", matrix[row][col]);
    }
    System.out.println();
}
```

Örnekler

1. **int[] {1, 5, 2, 2, 3, 4, 1}** dizisi içinde tekrarlı kayıtları silen programı yazalım.

2. **int[] {1, 2, 3}** dizisini tersine çevirerek yazdıran programı yazalım.

3. **"This was a String to reverse"** String ifadesini tersten yazdıran programı yazalım.

4. 4 öğrencinin bir dersten iki vize ve bir final notunun klavyeden girişi yapılarak, o derse ait, öğrencilerin not ortalamalarının hesaplanması ve sınıfın not ortalamasının hesaplanarak ekrana yazdırılması hedeflenmiştir.

5. Java 1D Array
   https://www.hackerrank.com/challenges/java-1d-array-introduction/problem?isFullScreen=true

6. Java Subarray
   https://www.hackerrank.com/challenges/java-negative-subarray/problem?isFullScreen=true

7. Best Time to Buy and Sell Stock
   https://leetcode.com/problems/best-time-to-buy-and-sell-stock/