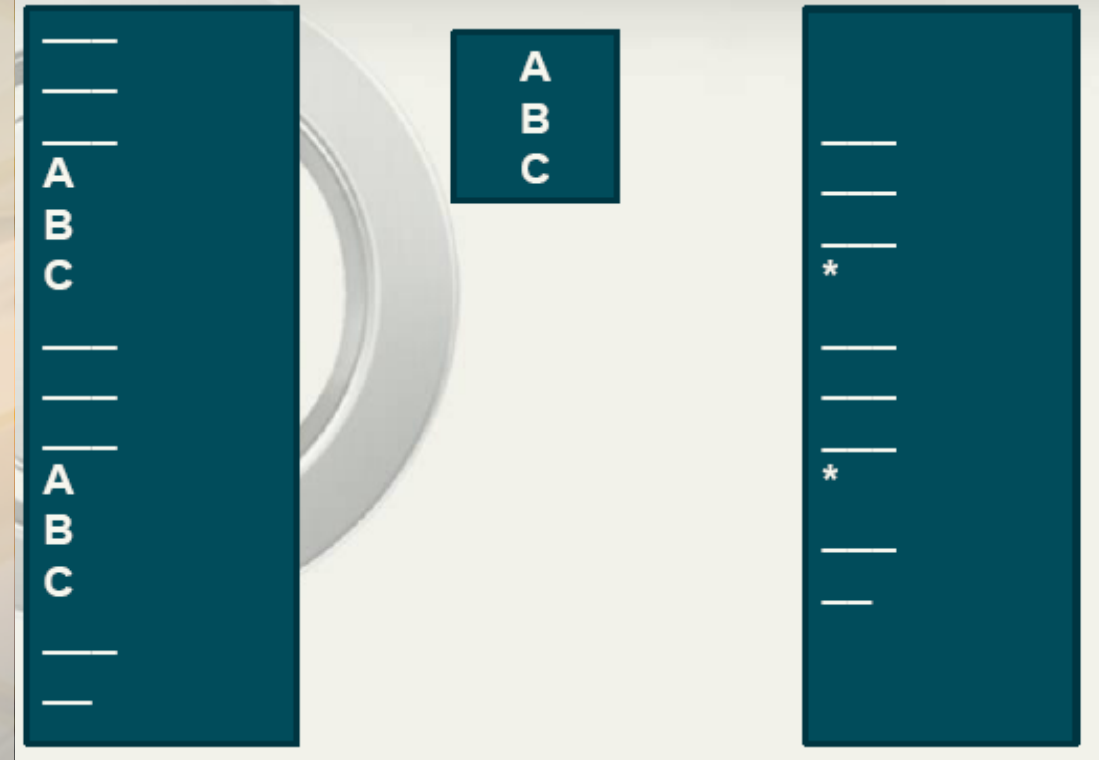


# FONKSİYONLAR

Function (Metotlar)

# FONKSİYON

- Bir program yazarken, tasarlarken belli kod bloklarını sık sık kullanırız. Bu kod bloklarını tekrar tekrar yazmak yerine bir defa yazıp o kodları çağırabiliriz.



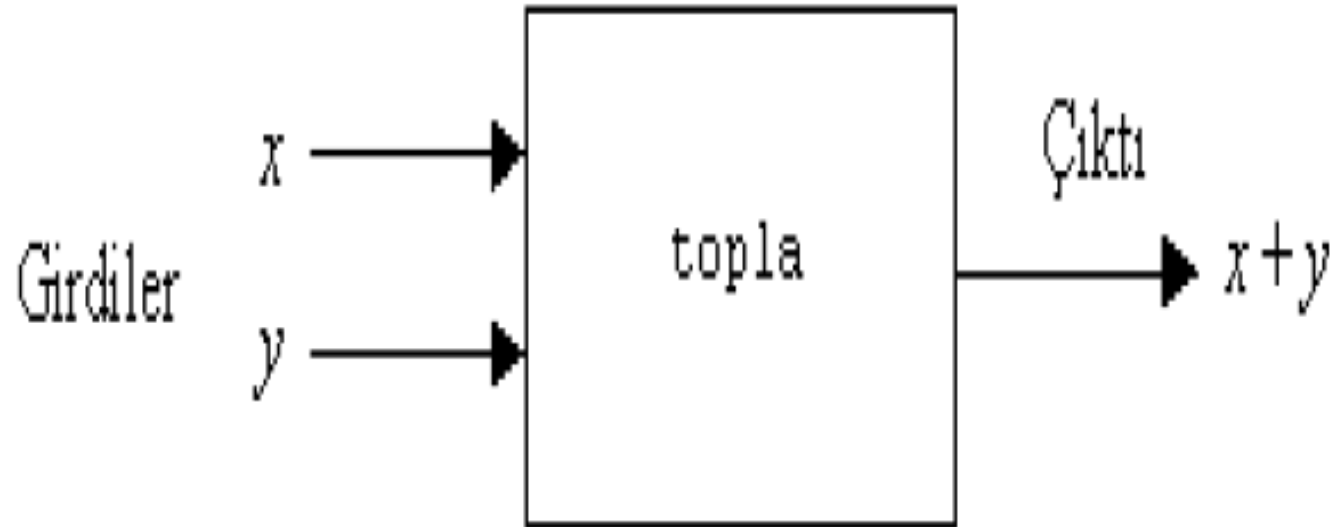
# FONKSİYON

Fonksiyon tipi: int

Fonksiyon adı : topla

parametreler : x ve y

geri dönüş değeri:  $x+y$





# FONKSİYON TANIMLAMA

Access Modifier + *Return Type* + *Method ismi* + *()* + { }

*public static FonksiyonTipi FonksiyonAdı(parametre listesi)*

{

*Yerel değişkenlerin bildirimi ...*

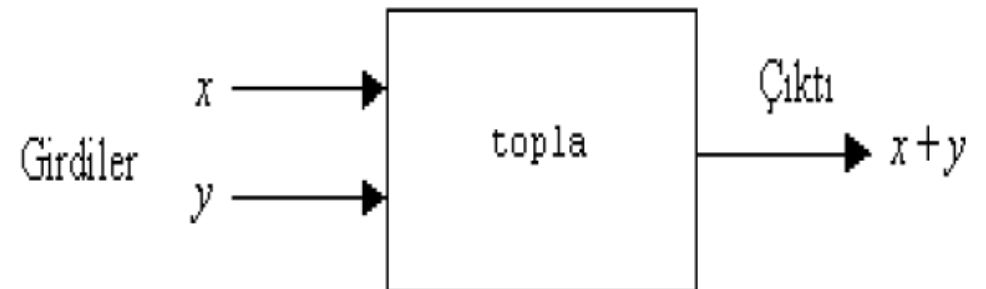
*fonksiyon içindeki deyimler veya diğer fonksiyonlar ...*

*return geri dönüş değeri;*

}

Fonksiyon tipi: int  
Fonksiyon adı : topla  
parametreler : x ve y  
geri dönüş değeri: x+y

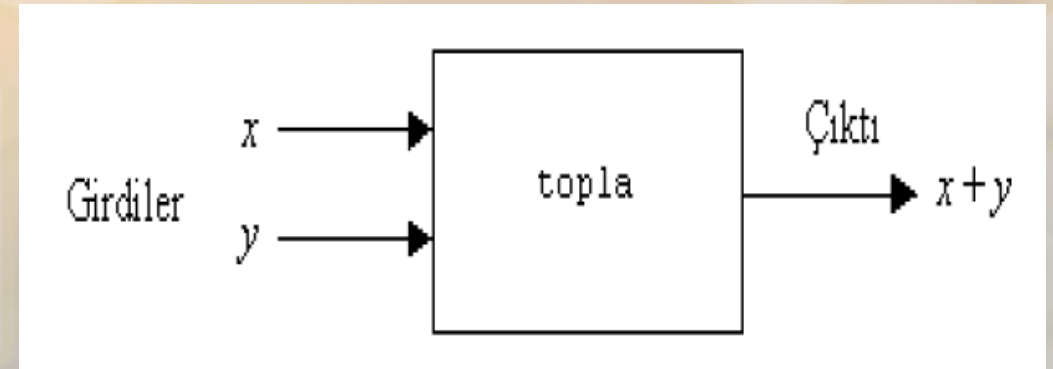
Öğretim Görevlisi Naciye MACİT SEZİKLİ



# FONKSİYON TANIMLAMA

```
public static int topla (int x, int y)  
{  
    int toplam=x+y;  
    return toplam;  
}
```

Fonksiyon tipi: *int*  
Fonksiyon adı : *topla*  
parametreler : *x* ve *y*  
geri dönüş değeri: *x+y*



# FONKSİYON TANIMLAMA

```
package d;|
public class D {
    public static void main(String[] args) {

    }
    public static int topla(int x, int y)
    {
        int toplam=x+y;
        return toplam;
    }
}
```

# FONKSİYON TANIMLAMA

```
public class D {  
    public static void main(String[] args) {  
        System.out.println(topla(3,4));  
    }  
    public static int topla(int x, int y)  
    {  
        int topla=x+y;  
        return topla;  
    }  
}
```

it - d (run) ×

run:

7

# Parametre Alan Geriye Değer Döndüren Fonksiyon

```
public class D {  
    public static void main(String[] args) {  
        System.out.println(topla(3,4));  
    }  
    public static int topla(int x, int y)  
    {  
        int topla=x+y;  
        return topla;  
    }  
}
```

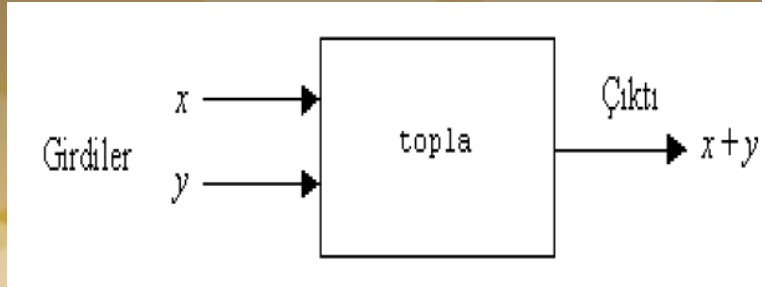
it - d (run) ×

run:

7

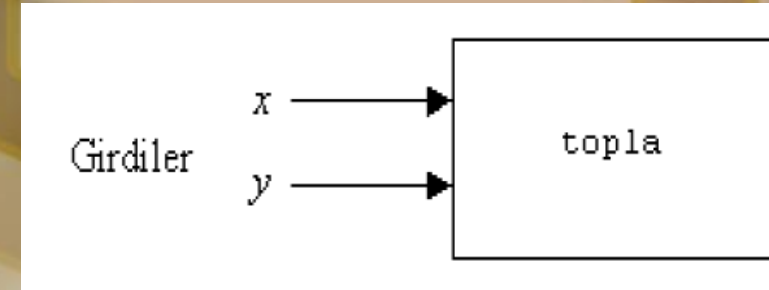


## Parametre Alan/ Geriye Değer Döndüren/



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

## Parametre Alan/ Geriye Değer Döndürmeyen/



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

# Parametre Alan Geriye Değer Döndürmeyen Fonksiyon

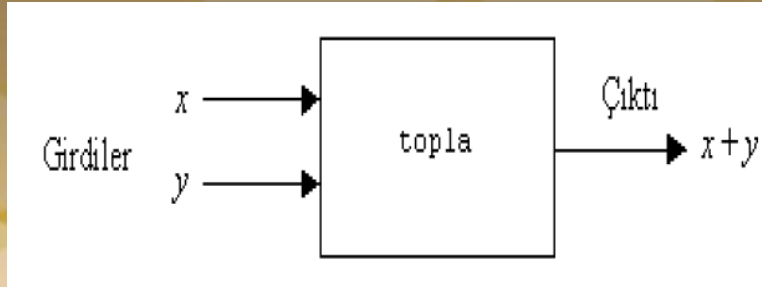
```
package d;  
public class D {  
    public static void main(String[] args) {  
        toplar(3,4);  
    }  
    public static void toplar(int x, int y)  
    {  
        int toplar=x+y;  
        System.out.println(toplar);  
    }  
}
```

ut - d (run) ×

run :

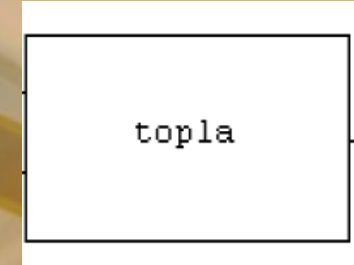
7

## Parametre Alan/ Geriye Değer Döndüren/



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

## Parametre Almadı Geriye Değer Döndürmedi /



```
void topla()
{
    int toplam=5+6;
    System.out.println(toplam);
}
```

# Parametre Almayan Geriye Değer Döndürmeyen Fonksiyon

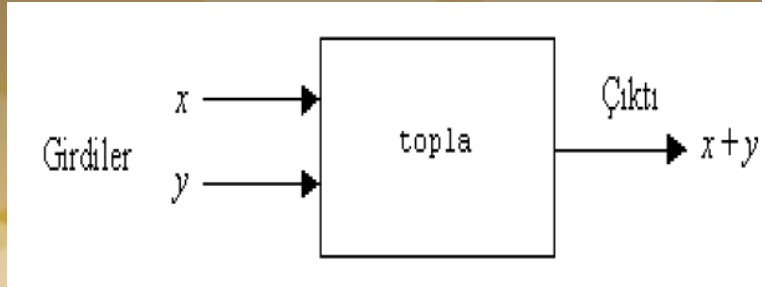
```
package yeni;

public class Yeni {
    public static void toplama()
    {
        int x=5,y=7;
        int toplam =x+y;
        System.out.println(toplam);
    }
    public static void main(String[] args) {
        toplama();
    }
}

<
yeni.Yeni > main >
ut - yeni (run) x
run :
12
```

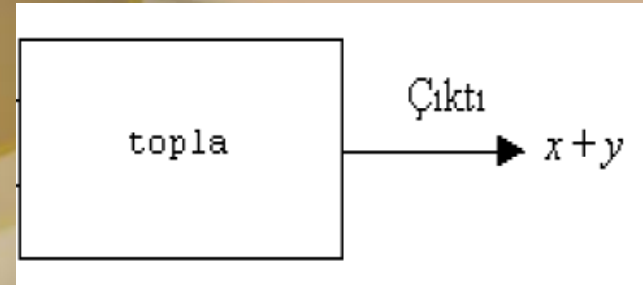


## Parametre Alan/ Geriye Değer Döndüren/



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

## Parametre Almayan/ Geriye Değer Döndüren/



```
int topla()
{
int x=5,y=7;
int toplam=x+y;
return toplam;
}
```

# Parametre Almayan Geriye Değer Döndüren Fonksiyon

```
package yeni;

public class Yeni {
    public static int toplama()
    {
        int x=5,y=7;
        int toplam =x+y;
        return toplam;
    }

    public static void main(String[] args) {
        System.out.println(toplama());
    }
}
```

< yeni.Yeni > main >

out - yeni (run) x

run:

12

## Parametre ALAN

## Parametre ALMAYAN

Geriye  
DEĞER  
DÖNDÜREN

```
public static int topla(int x, int y)
{
    int sonuc = x + y;
    return sonuc;
}
public static void Main(string[] args)
{
    int a =4;
    int b = 5;
    int c = topla(2,5);
    System.out.println(c);
}
```

```
public static int topla()
{
    int x = 4, y = 5;
    int sonuc = x + y;
    return sonuc;
}
public static void Main(string[] args)
{
    int a =4;
    int b = 5;
    int c = topla();
}
```

Geriye  
DEĞER  
DÖNDÜRMEYEN

```
public static void topla(int x, int y)
{
    int sonuc = x + y;
    System.out.println (sonuc);
}
public static void Main(string[] args)
{
    int a =4;
    int b = 5;
    topla(a,b);
}
```

```
public static void topla()
{
    int x = 4, y = 5;
    int sonuc = x + y;
    System.out.println (sonuc);
}
public static void Main(string[] args)
{
    int a =4;
    int b = 5;
    topla();
}
```

# Soru

Fonksiyona gönderilen vize final proje notunun ortalamasını hesaplayan programı tasarlayınız.

Vize %50

Final %30

Proje %20



# Soru

Fonksiyona gönderilen vize final notunun ortalamasını hesaplayan bir fonksiyon ve fonksiyona ortalama notunu gönderip harf notunu geri döndüren programı tasarlayınız.

## Soru

Klavyeden girilen sayının karesini alan programı fonksiyon ile tasarlayın.

# Soru

$$P(x) = a + bx + cx^2 + dx^3$$

- Klavyeden alınan a, b, c, d ve x değerlerini fonksiyona gönderip sonucu ekrana yazan fonksiyonu tasarlayınız.



# POLYMORPHİSM

## çok biçimlilik



# POLYMORPHISM

- 1- Method Overloading (static polymorphism)*
- 2- Method Overriding (dynamic polymorphism)*

*OOP'nin (Object Oriented Programming-Nesne yönelimli programlama) prensiplerinin biridir.*

*Polymorphism (çok biçimlilik) demektir.  
Bir metodu, istediğimiz sonuçları alacak şekilde, farklı farklı şekillerde çalıştırabilmek için kullanılır.*

# POLYMORPHISM

*Method Overloading (static polymorphism)  
Method Overriding (dynamic polymorphism)*

*OOP'nin (Object Oriented Programming-Nesne yönelimli programlama) prensiplerinin biridir.*

*Polymorphism (çok biçimlilik) demektir.*

*Bir method'u, istedigimiz sonuclari alacak sekilde, farkli farkli sekillerde calistirabilmek icin kullanilir.*

# METHOD OVERLOADING

(static polymorphism-Fonksiyonların Aşırı Yüklenmesi)

- Aynı isimde iki fonksiyon tanımlanabilir mi?
- Aynı işlevi yapacak fonksiyonların herbirine ayrı isim vermek ne tür sorunlar doğurur?
- İşlev – isim benzerliği olan metotları nasıl yöneteceğiz?



# METHOD OVERLOADING

(static polymorphism-Fonksiyonların Aşırı Yüklenmesi)

- Bu tür durumlarda aynı isimde ama farklı işlevlere sahip fonksiyonlar oluşturulabilir.
- İsimler aynı
- Parametre sayısı farklı olabilir
- Parametre data tipleri farklı olabilir
- Parametre data tipleri farklı ise yerleri değişebilir
- *return type'in, access modifier'in, static veya non-static olmanın hiçbir etkisi yoktur.*



# FONKSİYONLARIN YAPILARI

```
public static void main(String[] args) {  
    add(2, 3, 4);  
}  
  
public static void add(int a, int b) {  
    System.out.println(a + b);  
}  
  
public static void add(double a, double b) {  
    System.out.println(a + b);  
}  
  
public static void add(double a, int b) {  
    System.out.println(a + b);  
}  
  
public static void add(int a, double b) {  
    System.out.println(a + b);  
}  
  
public static void add(int a, int b, int c) {  
    System.out.println(a + b + c);  
}
```

# FONKSİYONLARIN YARATILMASI

```
public static void main(String[] args) {  
    add(2, 3.4);  
}  
  
public static void add(int a, int b) {  
    System.out.println(a + b);  
}  
  
public static void add(double a, double b) {  
    System.out.println(a + b);  
}  
  
public static void add(double a, int b) {  
    System.out.println(a + b);  
}  
  
/* public static void add(int a, double b) {  
    System.out.println(a + b);  
} */  
  
public static void add(int a, int b, int c) {  
    System.out.println(a + b + c);  
}
```

```
public static void main(String[] args) {  
    add(2, 3.4);  
}  
  
public static void add(int a, int b) {  
    System.out.println(a + b);  
}  
/* public static void add(double a, double b) {  
    System.out.println(a + b);  
} */  
  
public static void add(double a, int b) {  
    System.out.println(a + b);  
}  
/* public static void add(int a, double b) {  
    System.out.println(a + b);  
} */  
  
public static void add(int a, int b, int c) {  
    System.out.println(a + b + c);  
}
```

# Soru

- **Dörtgen** adında bir fonksiyon oluşturun
- İçerisine tek bir değer giderse karenin çevresi( $4 * x$ )
- İçerisine 2 değer giderse dikdörtgenin çevresi( $(x+y) * 2$ )
- İçerisine 4 değer giderse dörtgenin çevresini hesaplayın.  $(x+y+z+m)$



# VARARGS

# Varargs

*Bir methodun değişken sayıda argüman almasına izin veren bir özelliktir.*

*Varargs, "variable arguments" yani "değişken sayıda argüman"ın kısaltmasıdır.*

```
public static void main(String[] args) {  
    add(2, 4, 65, 90, 4, 2);  
    add(2, 4, 7);  
    add(5, 4);  
}
```

```
public static int add(int... a) {  
    int sum = 0;  
    for (int w : a) {  
        sum = sum + w;  
    }  
    return sum;  
}
```

# Varargs

*Varargs son parametre olmalıdır.*

```
public void exampleMethod(String name, int... numbers) {  
    // Gecerli  
}
```

varargs parameter must be the last parameter  
----  
(Alt-Enter shows hints)

```
public void exampleMethod(int... numbers, String name) {  
    // Geçersiz - Compile-time error  
}
```



# Varargs

**Varargs kullanımı isteğe bağlıdır:** Varargs ile metot çağrılırken hiç argüman da verilmeyebilir, tek bir argüman da verilebilir, birçok argüman da verilebilir. Eğer hiç argüman verilmezse, varargs parametresi boş bir dizi olarak alınır.

```
public static void main(String[] args) {  
    add();  
}  
  
public static int add(int... a) {  
    int sum = 0;  
    for (int w : a) {  
        sum = sum + w;  
    }  
    return sum;  
}
```

# Varargs

**Varargs ile Dizi Geçirme:** Varargs parametresine elle bir dizi de geçirebilirsiniz. Java, bunu otomatik olarak kabul eder.

```
public static void main(String[] args) {  
    int[] numArray = {1, 2, 3};  
    add(numArray);  
}  
  
public static int add(int... a) {  
    int sum = 0;  
    for (int w : a) {  
        sum = sum + w;  
    }  
    return sum;  
}
```

# Varargs

*Varargs, aşağıdaki kurallara tabidir:*

- 1- Birden fazla varargs kullanılamaz.*
- 2-Varargs parametre, her zaman son parametre olmalıdır.  
Yoksa veri tabanı sürekli veri kabul eder ve diğer parametreye geçemez.*
- 3- Varargs arka planda Array yapısını kullanır*
- 4- Varargs parametre, bir primitive veri tipi veya non-primitive(referans) olabilir.*

*Varargs, belirli bir veri tipinde esneklik sağlamak için güçlü bir araçtır ve çoklu parametrelili metotlar yazmayı oldukça kolaylaştırır.*



# Soru

Aşağıdaki adımları izleyerek bir Converter isimli Java sınıfı oluşturun. Bu sınıfta, farklı türlerde birim dönüşümleri gerçekleştiren metodları overload edeceksiniz.

Bu sınıfta, farklı türlerde birim dönüşümü yapan metodları overload edin. Overload Edilecek Metodlar: `convert(double kilometers)`: Kilometreyi mile çevirir. Dönüşüm formülü:  $\text{kilometers} * 0.621371$

`convert(int feet)`: Ayak uzunluğunu metreye çevirir. Dönüşüm formülü:  $\text{feet} * 0.3048$

`convert(double celsius, String type)`: Celsius'u Fahrenheit'e çevirir. Dönüşüm formülü:  $(\text{celsius} * 9/5) + 32$ . Bu metod sadece ikinci parametre olarak "fahrenheit" verilirse çağrılacak.

`convert(double kilograms, int precision)`: Kilogramı pound'a çevirir ve sonucu verilen ondalık hassasiyette döndürür. Dönüşüm formülü:  $\text{kilograms} * 2.20462$ . `precision` parametresi, sonucu kaç ondalık basamakla göstereceğinizi belirtir.

`main` metodunda bu metodları kullanarak farklı birim dönüşümleri yapın ve sonuçları ekrana yazdırın. Beklenen Çıktı Örneği: Kilometers to Miles: 6.21371

Feet to Meters: 3.048

Celsius to Fahrenheit: 98.6

Kilograms to Pounds (2 decimal places): 220.46

# Soru

Aşağıdaki adımları izleyerek bir Statistics isimli Java sınıfı oluşturun. Bu sınıfta, değişken sayıda sayıları işleyen metodlar tanımlayacaksınız.

Aşağıda belirtilen metodları varargs kullanarak yazın: findMax(int... numbers): Girilen sayılar arasındaki en büyük sayıyı bulur.

findMin(int... numbers): Girilen sayılar arasındaki en küçük sayıyı bulur.

calculateAverage(double... numbers): Girilen sayılar arasındaki ortalamayı hesaplar.

main metodunda bu metodları çağırarak farklı sayılar üzerinde işlemler yapın ve sonuçları ekrana yazdırın.



# FONKSİYONLARA DİZİLERİN GÖNDERİLMESİ



# DİZİ ELEMANINI FONKSİYONA GÖNDERMEK

```
package yeni;  
import java.util.Scanner;  
public class Yeni {  
    public static void hesapla(int x)  
    {  
        System.out.println(x*5);  
    }  
    public static void main(String[] args) {  
        int [] dizi= {3,2,11,33,13,4,2};  
        hesapla(dizi[6]);  
    }  
}
```

< *Yeni.Yeni* > *main* >  
- yeni (run) x

run :

10

# DİZİYİ FONKSİYONA GÖNDERMEK

```
package yeni;  
import java.util.Scanner;  
public class Yeni {  
    public static void hesapla(int [] gelendizi)  
    {  
        for (int i = 0; i < gelendizi.length; i++) {  
            System.out.print(gelendizi[i]+"-");  
        }  
    }  
    public static void main(String[] args) {  
        int [] dizi= {3,2,11,33,13,4,2};  
        hesapla(dizi);  
    }  
}
```

Yeni.Yeni > hesapla > for (int i = 0; i < gelendizi.length; i++) >

- yeni (run) x

run:

3-2-11-33-13-4-2-BUILD SUCCESSFUL (total time: 0 seconds)

Öğretim Görevlisi Naciye MACİT SEZİKELİ

# PASS BY VALUE

# Değişkeni FONKSİYONDA DEĞİŞTİRMEK

```
package yeni;  
import java.util.Scanner;  
public class Yeni {  
    public static void degistir(int gelen)  
    {  
        gelen=11;  
    }  
    public static void main(String[] args) {  
        int D= 3;  
        degistir(D);  
        System.out.println(D);  
    }  
}
```



# PASS BY VALUE

Metotlara parametre olarak verilen değişkenlerin kopyalarının alınarak işlenmesi anlamına gelir.

Yani, metot içinde yapılan değişiklikler orijinal değişkeni etkilemez, sadece kopya üzerinde yapılır. Java'da her şey pass by value ile aktarılır, ancak referans türleri için bu durum biraz kafa karıştırıcı olabilir.

# PASS BY VALUE

Yandaki örnekteki gibi bir metoda bir primitif veri tipi gönderildiğinde, bu değerin bir kopyası metotta kullanılır.

**Primitif tipler** (int, boolean, double, vb.) doğrudan değer olarak aktarılır.

```
package yeni;  
import java.util.Scanner;  
public class Yeni {  
    public static void degistir(int gelen)  
    {  
        gelen=11;  
    }  
    public static void main(String[] args) {  
        int D= 3;  
        degistir(D);  
        System.out.println(D);  
    }  
}
```

D nin içeriği değişmez.  
D etkinlenmez.

Kopyası olan gelen değişkeni değiştir.

# DİZİYİ FONKSİYONDA DEĞİŞTİRMEK

```
public static void main(String[] args) {  
    int[] numArray = {1, 2, 3};  
    degistir(numArray);  
    System.out.println(numArray[0]);  
    System.out.println(numArray[1]);  
    System.out.println(numArray[2]);  
}  
  
private static void degistir(int[] Arr) {  
    Arr[0]=8;  
    Arr[1]=5;  
}
```



# PASS BY VALUE

**Referans türleri** (nesneler) de pass by value ile aktarılır, ancak burada dikkat edilmesi gereken nokta, referansın kendisinin bir kopyasının aktarılmasıdır. Yani metot içinde bu referans üzerinden nesnenin içeriğine erişilebilir ve bu içerik değiştirilebilir, fakat referansın kendisi değiştirildiğinde orijinal nesne etkilenmez.



```
public static void main(String[] args) {  
    int[] dizi = {10}; // dizi bir adet sayı içeriyor  
    diziyiDegistir(dizi); // Dizinin içeriği değiştiriliyor  
    System.out.println("diziyiDegistir sonrası: " + dizi[0]);  
    // Çıktı: 20  
    referansiDegistir(dizi); // Referans başka bir diziyeye atanıyor  
    System.out.println("referansiDegistir sonrası: " + dizi[0]);  
    // Çıktı: 20 (değişmez)  
}  
  
public static void diziyiDegistir(int[] d) {  
    d[0] = 20; // Dizinin içeriği değişir (aynı nesneye erişiyoruz)  
}  
  
public static void referansiDegistir(int[] d) {  
    d = new int[1]; // Yeni bir diziyeye referans atanıyor  
    d[0]=30; // Bu değişiklik sadece bu metodun içindedir  
}
```

PASS

```
public static void main(String[] args) {  
    int[] dizi = {10};  
    // dizi bir adet sayı içeriyor  
    diziyiDegistir(dizi); // Dizinin içeriği değiştiriliyor  
    System.out.println("diziyiDegistir sonrası: " + dizi[0]);  
    // Çıktı: 20  
}
```

```
public static void diziyiDegistir(int[] d) {  
    d[0] = 20; // Dizinin içeriği değişir (aynı nesneye erişiyoruz)  
}
```

Adım adım ne oluyor?

- dizi adında bir referansımız var ve bu referans, bellekte bir int dizisini işaret ediyor.

Bu dizinin ilk (ve tek) elemanı başlangıçta 10 olarak ayarlanmış:

- **diziyiDegistir(dizi)** metodu çağrıldığında: dizi referansının bir **kopyası** metota geçer. Yani, hem main metodundaki dizi hem de diziyiDegistir metodundaki d değişkeni **aynı diziye işaret eder**.

Bu, sanki aynı eve giden iki farklı anahtar gibi düşünülebilir.

```
public static void diziyiDegistir(int[] d) { d[0] = 20; }
```

- Bu referans üzerinden **d[0] = 20;** satırı çalıştığında, dizinin ilk elemanı (dizi[0]) **20 olarak güncellenir**.
- Çünkü hem d hem de dizi **aynı diziye** gösteriyor. Bu yüzden içerik değişikliği dışarıya da yansır.

```
public static void main(String[] args) {  
    int[] dizi = {10};  
    // dizi bir adet sayı içeriyor  
    diziyiDegistir(dizi); // Dizinin içeriği değiştiriliyor  
    System.out.println("diziyiDegistir sonrası: " + dizi[0]);  
    // Çıktı: 20  
    referansiDegistir(dizi); // Referans başka bir diziye atanıyor  
    System.out.println("referansiDegistir sonrası: " + dizi[0]);  
    // Çıktı: 20 (değişmez)  
}
```

```
public static void referansiDegistir(int[] d) {  
    d = new int[1]; // Yeni bir diziye referans atanıyor  
    d[0]=30; // Bu değişiklik sadece bu metodun içindedir  
}
```

✗ referansiDegistir(dizi) metodu çağrıldığında:

- Yine dizi referansının bir **kopyası** metota geçer.

```
public static void referansiDegistir(int[] d) { d = new int[]{30}; }
```

- Ancak bu metotta, d referansı yeni bir diziye (**new int[]{30}**) yönlendirilir. Bu değişiklik **sadece metot içindedir**, çünkü dışarıdaki dizi hala eski diziye işaret eder.

- Bu yüzden bu metotta yapılan referans değişikliği, main metodundaki dizi değişkenini **etkilemez**.