

# NESNEYE YÖNELİK PROGRAMLAMA

*Final, Statik değişkenler, metotlar ve sınıflar  
Üretici Metotlar, Eleman sınıflar*

*Emir Öztürk*

## FINAL

- Sabit
- Değer atama
  - Tanımlama anında
  - Yapıcı çağırıldığında

Final ile kullanılan değişkenlerin tanım anında ilk değer ataması gerçekleştirilir. Daha sonra bu değişkenlerin değerleri değiştirilemez. Ayrıca final tanımlanmış bir değişken yapıcı çağırıldığında da ilk değer ataması gerçekleştirilebilir.

## FINAL

```
class Sinif{
    final private int cokOnemliDeger = 42;
    Sinif(){
    }
}

class Sinif{
    final private int cokOnemliDeger;
    Sinif(){
        cokOnemliDeger = 42;
    }
}
```

```
class Sinif{
    final private int cokOnemliDeger;
    Sinif(){
        cokOnemliDeger = 42;
    }
    int getCokOnemliDeger(){
        return cokOnemliDeger;
    }
    void setCokOnemliDeger(int deger){
        cokOnemliDeger = deger;
    }
}

public class Main {
    public static void main(String[] args) {
        Sinif s = new Sinif();
        System.out.println(s.getCokOnemliDeger());

        s.setCokOnemliDeger(59);
    }
}
```

İlk değer ataması değişken bildiriminin yapıldığı yerde yapılabileceği gibi yapıcı fonksiyon içerisinde de gerçekleştirilebilir. final ile tanımlanmış bir değer bir getter ile okunabilir ve erişilebilir. Değer değiştirilemeyeceği için set edilemeyecektir.

## STATIC

- Nesne değerleri
- Sınıf değerleri
  - Nesnelerden erişim
  - Her nesne için aynı değer

Bir sınıftan bir nesne oluşturulduğunda her nesne, tanımlanmış özellikleri için kendine ait değerler içerir.

static tanımında ise değer nesnelere değil sınıfa aittir. Bu değere her nesne tarafından erişilebilir.

## STATİK ALANLAR

```
class Kisi{
    static int siradakiID=0;
    private int kisiID;
    Kisi(){
        kisiID = siradakiID++;
    }
    int getKisiID(){
        return kisiID;
    }
}

public class Main {
    public static void main(String[] args) {
        Kisi k = new Kisi();
        System.out.println(k.getKisiID());

        Kisi k1 = new Kisi();
        System.out.println(k1.getKisiID());

        Kisi k2 = new Kisi();
        System.out.println(k2.getKisiID());

        int statikDeger = Kisi.siradakiID;
        System.out.println(statikDeger);
    }
}
```

Ekran Çıktısı

0
1
2
3

Diagram illustrating the static field `siradakiID` and instance fields `kisiID` for objects `k`, `k1`, and `k2`. The static field `siradakiID` is shared and increments from 0 to 3. Each instance has its own `kisiID` field, which is assigned the value of `siradakiID` at the time of creation.

Sınıftan oluşturulmuş her nesne `siradakiID` değerini arttırmaktadır. Her nesnenin `kisiID`'si static olmadığından kendine ait olurken, static olan `siradakiID` her nesne çağırımında bir artmakta ve tüm nesneler için aynı değer geçerli olmaktadır.

## STATİK METOTLAR

- Nesneden bağımsız parametreler - `Math.pow()`
- `this` anahtar kelimesi
- Sınıftan çağırım

Statik metotlar nesne ile ilgili olmayan işlemlerin yapılmasında tercih edilirler. Örneğin `Math.pow` metodu sınıfın adı ile çağırılır ve statiktir. Bu metot verilen sayının üssünü almak için kullanılır.

Bu metotta herhangi bir `Math` nesnesi kullanılmamaktadır. Statik metotlar `this` anahtar kelimesinin kullanılmadığı metotlar olarak düşünülebilir.

## STATİK DEĞİŞKEN VE METOTLARA ERİŞİM

```
class cokOnemliSınıf{
    final static int cokOnemliDeger=42;
}

public class Main {
    public static void main(String[] args) {
        cokOnemliSınıf cos = new cokOnemliSınıf();

        System.out.println(cos.cokOnemliDeger);

        System.out.println(cokOnemliSınıf.cokOnemliDeger);
    }
}

class cokOnemliSınıf{
    final static int cokOnemliDeger=42;
    static void cokOnemliMetot(){
    }
}

public class Main {
    public static void main(String[] args) {
        int deger = cokOnemliSınıf.cokOnemliDeger;
        cokOnemliSınıf.cokOnemliMetot();
    }
}
```

Statik değişkenlere ve metotlara sınıfın adı ile erişmek mümkündür. Sınıftan bir nesne oluşturmadan sınıfın adı ve . kullanılarak değerlere erişilebilir. Örneğin System.out sınıfı ve Math.PI değişkeni buna örnek verilebilmektedir. Ayrıca tercih edilmese de statik değişken ve metotlara sınıftan oluşturulmuş nesneler üzerinden de erişilebilir.

## STATİK ALAN ERİŞİMİ

```
class cokOnemliSınıf{
    int statikOlmayanDeger;

    static int cokOnemliDeger=42;

    cokOnemliSınıf(){
        statikOlmayanDeger = 5;
        cokOnemliDeger = 23;
    }

    static void cokOnemliMetot(){
        statikOlmayanDeger = 5;

        cokOnemliDeger = 23;
    }
}
```

Diagram illustrating static and non-static field access. The top row shows 'cokOnemliDeger' (static) and 'cokOnemliMetot()' (static). The bottom row shows three instances: 'Nesne 1', 'Nesne 2', and 'Nesne 3', each with its own 'statikOlmayanDeger' (non-static). Green lines indicate that static fields and methods can be accessed from any instance. Red lines indicate that non-static fields can only be accessed from the instance they belong to.

Statik olan alanlara sınıf içerisinde erişilebilmesine rağmen statik olmayan alanlara statik metotlar erişilememektedir.

Bunun sebebi statik bir alanın sınıftan üretilen her nesne için ortak olması ve bu ortak alana erişimin mümkün olmasıdır. Statik olarak sınıftan üretilen her nesne için ortak tanımlanmış bir metot ise nesnelere ait olan alanlara erişilemeyecektir.

## STATİK - FINAL STATİK

- Statik
- Final Statik
  - final static System.out
  - System.out = new PrintStream();

```
/*
 * See the @code println methods in class @code PrintWriter.
 */
@code java.io.PrintWriter.println()
 * @code java.io.PrintWriter.println(int)
 * @code java.io.PrintWriter.println(char)
 * @code java.io.PrintWriter.println(char[])
 * @code java.io.PrintWriter.println(double)
 * @code java.io.PrintWriter.println(float)
 * @code java.io.PrintWriter.println(int)
 * @code java.io.PrintWriter.println(long)
 * @code java.io.PrintWriter.println(java.lang.Object)
 * @code java.io.PrintWriter.println(java.lang.String)
 */
public static final PrintStream out = null;

/*
 * The "standard" error output stream. This stream is already
 * open and ready to accept output data.
 *
 *
 * Typically this stream corresponds to display output or another
 * output destination specified by the host environment or user. By
 * convention, this output stream is used to display error messages
 * or other information that should come to the immediate attention
 */
```

Statik değişkenlerin kullanımı çok yaygın değildir. Fakat final statik değişkenler oldukça fazla kullanılır.

Örneğin System sınıfı altında PrintStream türünden olan out özelliği final static olarak tanımlanmıştır.

Bu sayede hem out bir kere atandıktan sonra değiştirilememekte hem de new kelimesi ile bir örneğinin alınmasına gerek kalmamaktadır.

### MAIN METODU

- Program çalıştırıldığında çalışır
- Program başlangıcı
  - Herhangi bir nesne tanımlanmaz
- JVM başlangıcı
- Başlangıç parametresi

Yazılan bir java programı çalıştırıldığında JVM çalıştırılır. JVM hiçbir nesne üretilmediği başlangıç durumunda main metodunu çağırmalıdır. Nesne üretilmeden bir metodun çağırılması için bu metodun static olması gerekmektedir. JVM çalışma parametresi olarak bir class ismi alır (örn. java main.class). Daha sonra bu classtan bir örnek oluşturmadan static olan main metodunu arar. Bulunan main metodunun çalıştırılması işlemi gerçekleştirilir.

### ÜRETİCİ METOTLAR (FACTORY METHODS)

- Sınıftan nesne üreten metotlar
- Nesne içerisinde tanımlanırlar
- İlk değerin belli olması
- ~~Yapıcı~~
- İsimlendirilmiş yapıcı

Üretici metotlar kendi türünden nesneleri döndüren metotlardır. Bu metotlar kullanılarak yapıcı çağırılmadan nesne örneği almak mümkündür. Kullanımının bir sebebi de açıkça ilk alınacak değerin tanımlanmasıdır. Üretici metotların bir avantajı da isimlendirilmiş yapıcı gibi davranabilmeleridir. Böylece birden fazla farklı parametre alan yapıcılarda ne amaçla aldığı kod içerisinde belirtilebilmiş olur.

Ayrıca bazı durumlarda aynı türden parametre alan yapıcı ihtiyacı da duyulabilmektedir.

Örneğin veritabanından, internet adresinden veya dosyadan veri okuyacak bir sınıfın alabileceği string parametre yapıcıda değişmemektedir. Bu durumda hem hangi yapıcının çağırılacağını bilmesi, hem de kod içerisinde açıklayıcı olması amacıyla üretici metotlar kullanılabilir.

## ÜRETİCİ METOTLAR (FACTORY METHODS)

```
public class Main {
    public static void main(String[] args) {

        LocalDate x = LocalDate.now();

        LocalDate y = new LocalDate();
    }
}
```

'LocalDate(int, int, int)' has private access in 'java.time.LocalDate'

Örneğin LocalDate sınıfı bir yapıcı içerse de bu yapıcı private olarak tanımlanmıştır. Bu sebeple yapıcı ile boş bir nesne oluşturulamaz. LocalDate sınıfının now metodu ile içerisinde o anki tarih saati içeren bir tarih nesnesi döndürülür. Yapıcı kullanmak yerine üretici metot kullanılmış olur.

## ÜRETİCİ METOTLAR (FACTORY METHODS)

```
enum ResimTuru {
    Jpeg, Png, Gif, Webp
}

class Resim {
    private ResimTuru tur;
    private Resim(ResimTuru tur) {
        this.tur = tur;
    }
    static Resim jpeg() {
        return new Resim(ResimTuru.Jpeg);
    }
    static Resim png() {
        return new Resim(ResimTuru.Png);
    }
    static Resim gif() {
        return new Resim(ResimTuru.Gif);
    }
    static Resim webp() {
        return new Resim(ResimTuru.Webp);
    }
    byte[] Oku() {
        byte[] okunan = new byte[100];
        if(tur == ResimTuru.Jpeg) okunan = new byte[100]; //JPEG OKUMA İZLENLERİ...
        else if(tur == ResimTuru.Png) okunan = new byte[100]; //PNG OKUMA İZLENLERİ...
        else if(tur == ResimTuru.Gif) okunan = new byte[100]; //GIF OKUMA İZLENLERİ...
        else if(tur == ResimTuru.Webp) okunan = new byte[100]; //WEBP OKUMA İZLENLERİ...
        return okunan;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Resim jpegResim = Resim.jpeg();
        byte[] okunan = jpegResim.Oku();
    }
}
```

## Üretici Metodun yazılışı

## THIS ANAHTAR KELİMESİ

- Statik olmayan alanlar
- Sınıfın adresi
- Seçimlik

Statik olmayan alanlarda sınıfın adresini işaret eden anahtar kelimedir. Bir sınıfın alanlarına erişmek için kullanılabilir. Aynı isimde başka bir lokal değişken tanımlanmadığı takdirde java için kullanılması zorunlu değildir

### THIS ANAHTAR KELİMESİ

```
class Resim{
    byte deger;
    Resim(byte deger){
        deger = deger;
        this
    }
    static
    this.deger = deger;
}

class Resim2{
    byte deger;
    Resim2(byte deger){
        this.deger = deger;
    }
    static void degerDegistir(byte deger){
        this.deger = deger;
    }
}
```

'net.emirozturk.Resim2.this' cannot be referenced from a static context

Make 'degerDegistir' not static

deger değişkeni aynı isimde ise this kullanılması gereklidir.  
statik metotlarda this ifadesi kullanılamaz

### İÇ İÇE SINIFLAR (MEMBER CLASSES)

```
class Sinif1{
    int deger;
}

class Sinif2{
    Sinif1 degisken1;
}

class Sinif3{
    Sinif2 degisken2;
}

public class Main {
    public static void main(String[] args) {
        Sinif3 s3 = new Sinif3();
        int sonuc = s3.degisken2.degisken1.deger;
    }
}
```

Bir sınıftan oluşturulmuş nesneler başka bir sınıf için özellik olarak kullanılabilirler.

Bu işlem iç içe istenildiği kadar gerçekleştirilebilir.

Bu şekilde bir sınıf altında bir diğer sınıfın alan olarak kullanılmasına “eleman sınıf” adı verilir.