

NESNEYE YÖNELİK PROGRAMLAMA

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

Emir Öztürk

YAPICILAR

- Constructor
- Sınıf ismi ile aynı isim
 - Bazı diller için farklı
 - Sub new
 - __init__()
- Dönüş değeri bulunmaz (Değer döndürmez)
- Varsayılan olarak hiç parametre almayan bir yapıcı
- İlk yapıcı yazıldığında varsayılan kullanılmaz
- Birden fazla yapıcı yazılabilir

YAPICILAR - VARSAYILAN YAPICI


```
public class Sinif {  
    int alan1; not  
    String alan2;  
    double alan3;  
}
```



```
public static void main(String[] args) {  
    Sinif sinif = new Sinif();  
}
```

YAPICILAR - İLK EKLENEN YAPICI


```
public class Sinif {  
    int alan1; no usage  
    String alan2; no usage  
    double alan3; no usage  
    Sinif() { 1 usage  
    }  
}
```



```
public static void main(String[] args) {  
    Sinif sinif = new Sinif();  
}
```

YAPICILAR - VARSAYILAN YAPICININ OLMAMASI

```
public class Sinif { 2 usage  
    int alan1; 1 usage  
    String alan2; no usage  
    double alan3; no usage  
    Sinif(int alan1) { 1 usage  
        this.alan1=alan1;  
    }  
}
```



```
public class Main {  
    public static void main(String[] args) {  
        Sinif sinif = new Sinif();  
    }  
}
```

Expected 1 argument but found 0
Remove 1st parameter from constructor

```
@Sinif  
@Contract(pure = true) >  
Sinif(  
    int alan1  
)  
>  
yapici
```

YAPICILAR - BOŞ YAPICI EKLENMESİ

```
public class Sinif { 2 usage  
    int alan1; 1 usage  
    String alan2; no usage  
    double alan3; no usage  
    Sinif() { 1 usage  
    }  
    Sinif(int alan1) { no usage  
        this.alan1=alan1;  
    }  
}
```



```
public class Main {  
    public static void main(String[] args) {  
        Sinif sinif = new Sinif();  
        Sinif sinif1 = new Sinif(alan1);  
    }  
}
```

YAPICILAR

- Bir yapıcılığı birden fazla yazmak için farklı olması gereken alanlar bulunur
 - Parametre sayısı
 - Parametre sırası
 - Parametre türü

YAPICILAR – BIRDEN FAZLA YAZIM

```
public class Sinif { 4 usages 1 related
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    Sinif(){ 1 usage
    }
    Sinif(int alan1){ no usages 1 related
        this.alan1=alan1;
    }
    Sinif(int alan2){ no usages 1 related
        'Sinif(int)' is already defined in 'Sinif'
    }
}
```

YAPICILAR – BIRDEN FAZLA YAZIM

```
public class Sinif { 4 usages 1 related
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    Sinif(){ 1 usage
    }
    Sinif(int alan1){ no usages 1 related
        this.alan1=alan1;
    }
    Sinif(int alan2){ no usages 1 related
        'Sinif(int)' is already defined in 'Sinif'
    }
}
```

```
public class Sinif { 4 usages 1 related
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    Sinif(){ 1 usage
    }
    Sinif(int alan1){ 1 usage
        this.alan1=alan1;
    }
    Sinif(double alan2){
    }
}
```

YAPICILAR - FARKLI PARAMETRELERİN AYNI TÜRDE OLUŞU

```
public class Sinif { 4 usages 1 related problem
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    int alan4; no usages
    Sinif(){ 1 usage

    }

    Sinif(int alan1,String alan2){ no usages
        this.alan1=alan1;
    }

    Sinif(int alan4,String alan2){ no usages
        //String, String' is already defined in 'Sinif'
    }
}

@ Sinif
@Construct(pure = true) >
Sinif
    int alan1,
    String alan2
>
> yapici
```

YAPICILAR-PARAMETRE SIRASI

```
public class Sinif { 4 usages 1 related problem
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    int alan4; no usages
    Sinif(){ 1 usage 1 related problem

    }

    Sinif(int alan1,String alan2){
        this.alan1=alan1;
    }

    //Change signature
    Sinif(String alan2,int alan4){

    }
}
```

YAPICILAR - FARKLI TÜR VE SAYI

```
public class Sinif { 4 usages
    int alan1; 1 usage
    String alan2; no usages
    double alan3; no usages
    int alan4; no usages
    Sinif(){ 1 usage

    }

    Sinif(int alan1,String alan2,double alan3){
        this.alan1=alan1;
    }

    Sinif(int alan1,String alan2){ no usages

    }

    Sinif(int alan1,double alan3){ no usages

    }
}
```

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 çağırılacağına bilinmesi, 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;
public:
    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 İZLENİMLERİ*/}
        else if(tur == ResimTuru.Png) {okunan = new byte[100]; /*PNG OKUMA İZLENİMLERİ*/}
        else if(tur == ResimTuru.Gif) {okunan = new byte[100]; /*GIF OKUMA İZLENİMLERİ*/}
        else if(tur == ResimTuru.Webp) {okunan = new byte[100]; /*WEBP OKUMA İZLENİMLERİ*/}
        return okunan;
    }
};

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

Üretici Metodun yazılışı

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.

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;
    }
}

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

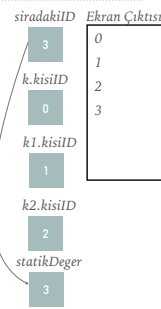
```
class Kisi{
    static int siradakiID=0;
    private int kisiID;
    Kisi(){
        kisiID = siradakiID++;
    }
    int getKisiID(){
        return kisiID;
    }
}

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);
    }
}
```



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 erişebilmek mümkündür.

STATİK ALAN ERİŞİMİ

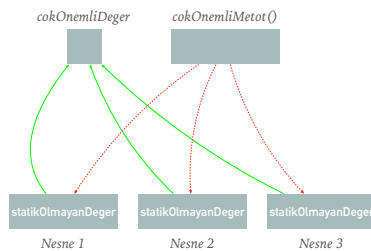
```
class cokOnemliSınıf{
    int statikOlmayanDeger;

    static int cokOnemliDeger=42;

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

    static void cokOnemliMetot(){
        statikOlmayanDeger = 5;

        cokOnemliDeger = 23;
    }
}
```



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

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 erişemeyecektir.

STATİK - FINAL STATİK

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

```
// -
// See the (code println) methods in class (code PrintStream).
//
// Done java.io.PrintStream.println()
// Done java.io.PrintStream.println(boolean)
// Done java.io.PrintStream.println(char)
// Done java.io.PrintStream.println(char[])
// Done java.io.PrintStream.println(double)
// Done java.io.PrintStream.println(float)
// Done java.io.PrintStream.println(int)
// Done java.io.PrintStream.println(long)
// Done java.io.PrintStream.println(java.lang.Object)
// Done java.io.PrintStream.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.

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

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