NESNEYE YÖNELİK PROGRAMLAMA

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

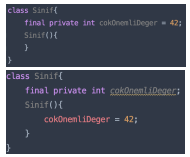
Emir Öztürk

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



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

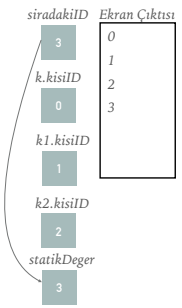
- Nesne değerleri
- Sınıf değerleri
  - Nesnelerden erişim
  - Her nesne için aynı değer

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

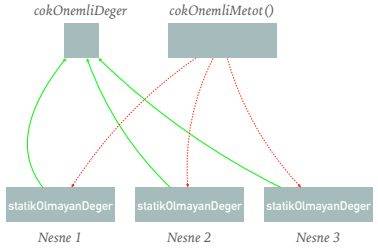
```java
class cokOnemliSinif{
    final static int cokOnemliDeger=42;
}
public class Main {
    public static void main(String[] args) {
        cokOnemliSinif cos = new cokOnemliSinif();

        System.out.println(cos.cokOnemliDeger);

        System.out.println(cokOnemliSinif.cokOnemliDeger);
    }
}
```

```java
class cokOnemliSinif{
    final static int cokOnemliDeger=42;
    static void cokOnemliMetot(){

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

```java
class cokOnemliSinif{
    int statikOlmayanDeger;

    static int cokOnemliDeger=42;

    cokOnemliSinif(){
        statikOlmayanDeger = 5;

        cokOnemliDeger = 23;
    }

    static void cokOnemliMetot(){
        statikOlmayanDeger = 5;

        cokOnemliDeger = 23;
    }
}
```



Statik olan alanlara sınıf içerisinden 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();~~

```java
/**
 * <p>
 * See the {@code println} methods in class {@code PrintStream}.
 *
 * @see     java.io.PrintStream#println()
 * @see     java.io.PrintStream#println(boolean)
 * @see     java.io.PrintStream#println(char)
 * @see     java.io.PrintStream#println(char[])
 * @see     java.io.PrintStream#println(double)
 * @see     java.io.PrintStream#println(float)
 * @see     java.io.PrintStream#println(int)
 * @see     java.io.PrintStream#println(long)
 * @see     java.io.PrintStream#println(java.lang.Object)
 * @see     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.
 * <p>
 * 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.

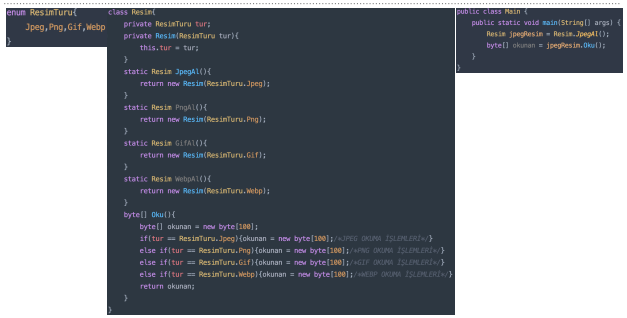
Ayrıca bazı durumlarda aynı türden parametre alan yapıcı ihtiyacı da duyulabilmektedir.

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

ÜRETİCİ METOTLAR (FACTORY METHODS)

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)



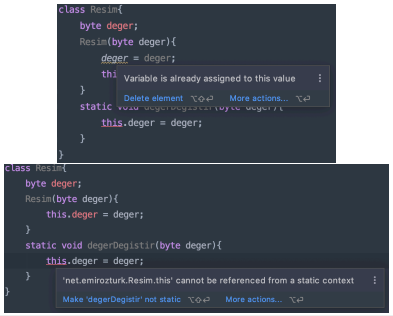
Üretici Metodun yazılışı

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



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 verililir.
