

NESNEYE YÖNELİK PROGRAMLAMA

Arayüzler ve Paketler

Emir Öztürk

ARAYÜZLER

- Sınıfın yapısı
- Soyut
- Ne / Nasıl
- Örnek oluşturma
- Metot gövdesi (Java 8 sonrası)

Arayüzler bir sınıfın yapısının nasıl uygulanacağından soyut bir şekilde net olarak ayrılması için kullanılabilirler.

Bir sınıfın ne yapacağını tanımlarlar fakat görevini nasıl yapacağını tanımlamazlar.

Bir arayüzden bir örnek (instance) oluşturulmaz.

Arayüz içerisinde tanımlanan metotların yalnızca tanımları (prototipleri) bulunur ve metot gövdesi içermezler.

Java 8 sonrası arayüzlerde varsayılan metot içeriği de tanımlanabilmektedir.

ARAYÜZLER

- Bir arayüz birden fazla sınıf
- Bir sınıf birden fazla arayüz

Bir arayüzü kullanan birden fazla sınıf olabileceği gibi, bir sınıf birden fazla arayüzü de içerebilir. Çoklu kalıtım konusunun çözülmesi için bir sınıfın birden fazla arayüzü içermesi sağlanabilir.

ARAYÜZLER

- Arayüzün adı
- Arayüzün metotları
- Sınıfın içermesi
 - Metot içeriği (Java 8 sonrası)

Bir arayüzün tanımlanmasından sonra o arayüzü bir sınıfın kullanabilmesi için, o sınıfın arayüzde tanımlanmış tüm metotların bir implementasyonunu içermesi gerekmektedir. Aksi takdirde bu sınıf bu arayüzü içermeyecek ve kod derlenmeyecektir. Arayüzlerde yalnızca metodun başlığı (döndürdüğü değer, ismi ve argümanları) bulunduğu için, metodun implementasyonu arayüzü içeren sınıflar içerisinde yapılmalıdır.

ARAYÜZLER

```
ErişimBelirleyicisi interface Arayuz{  
    DönüşTürü metot1();  
    DönüşTürü metot2(Tür değişkenAdı);  
    Tür degisken = Deger;  
    Tür degisken2 = Deger2;  
}
```

Arayüz tanımlanırken sınıflarda olduğu gibi Erişim belirleyicisi ile kullanılabilirler. Arayüzler metotların başlıklarını içerirler ama içeriklerini içermezler. Arayüzler içerisinde değişken tanımlanacaksa değerlerinin verilmesi gerekmektedir.

ARAYÜZLER

```
interface Arayuz{  
    void metot1();  
    int metot2(int a);  
    int degisken = 3;  
    double deger;  
}  
public class {  
    Variable 'deger' might not have been initialized  
    Initialize variable 'deger' More actions...
```

```
interface Arayuz{  
    void metot1();  
    int metot2(int a);  
    int degisken = 3;  
    double deger = 5.0;  
}
```

İlk değeri verilmemiş değişkenler hata verdiği için arayüzlerde ilk değişken tanımı gereklidir.

ARAYÜZLER

- Sınıfın kullanması
 - implements
- Arayüz metotları
- Sınıfın içerisindeki metotlar

Yazılan bir arayüzü bir sınıfın kullanabilmesi için implements anahtar kelimesi kullanılır.

Bir sınıf, arayüz metotlarının tamamını içermelidir fakat sınıf sadece arayüz metotlarını içermek zorunda değildir.

Sınıfta arayüz metotlarından farklı olarak kendine ait metotlar da bulunabilir.

ARAYÜZLER

```
interface IErisilebilir{
    int ninciEleman(int n);
}
interface IGezilebilir{
    int siradakiEleman();
}
```

```
class Liste implements IErisilebilir{
    ArrayList<Integer> liste;
    Liste(){
        liste = new Arr
    }
}
```



```
class Liste implements IErisilebilir{
    ArrayList<Integer> liste;
    Liste(){
        liste = new ArrayList<Integer>();
    }
    public int ninciEleman(int n) {
        return 0;
    }
}
```

IErisilebilir arayüzünü içeren bir Liste sınıfında arayüz metotları içerilmiyorsa hata verecektir.

Bu metotların yazılması zorunludur.

Bu metotlar dışında sınıf başka metotlar da içerebilir. Örneğin Liste sınıfının kendine ait bir yapıcısı bulunmaktadır.

ARAYÜZLER

```
interface IErisilebilir{
    int ninciEleman(int n);
}
interface IGezilebilir{
    int siradakiEleman();
}
class Liste implements IErisilebilir, IGezilebilir{
    ArrayList<Integer> liste;
    int siradaki = 0;
    Liste(){
        liste = new ArrayList<Integer>();
    }
    public int ninciEleman(int n){
        return liste.get(n);
    }
    public int siradakiEleman(){
        return liste.get(siradaki);
    }
}
```

Bir sınıf birden fazla arayüzü içerebilir. Örneğin hem IErisilebilir arayüzü hem de IGezilebilir arayüzünü içeren bir sınıf her iki arayüzün de metotlarını içermelidir. Birden fazla arayüzün kullanılması için implements kelimesinden sonra arayüzler arasına virgül konulmaktadır.

ARAYÜZLER

```
class Liste implements IErisilebilir{
    ArrayList<Integer> liste;
    int siradaki = 0;
    Liste(){
        liste = new ArrayList<Integer>();
    }

    public int ninciEleman(int n) {
        return liste.get(n);
    }
    public int boyutVer(){
        return liste.size();
    }
}
```

```
List<Integer> liste = new ArrayList<Integer>();
```

```
public class Main {
    public static void main(String[] args) {
        IErisilebilir nesne = new Liste();
        nesne.ninciEleman();
        nesne.boyutVer();
    }
}
```

Cannot resolve method 'boyutVer' in 'IErisilebilir'
Cast qualifier to 'net.emrorturk.Liste' or 'List' More actions...

Bir sınıfın tanımı miras alınan bir üst sınıf ile yapılabildiği gibi arayüzlerle de yapılabilmektedir.

Metot çağırımı sırasında bu arayüzler çalışma anında metodu çağırın sınıfı hesaplayacaklardır.

Arayüzle çağırım yapıldığında sınıfın arayüz dışındaki metotlarına erişilemez.

ARAYÜZLER

```
interface IErisilebilir{
    int ninciEleman(int n);

    static int metot(int statik) {
        return statik*2;
    }
}

public class Main {

    public static void main(String[] args) {
        IErisilebilir.metot( statik: 3512);
    }
}
```

Arayüzler içerisinde statik metotlar da tanımlanabilmektedirler.

Statik metotların metot gövdesini de içermeleri gerekmektedir.

Statik sınıf metotlarında olduğu gibi statik metotlar arayüz ismi kullanılarak çağırılabilirler.

PAKETLER

- Sınıfları bir arada tutmak
- Sınıf isimlerinin ayrımı
- Hiyerarşik

Paketler sınıfların bir araya getirilip organize edilmesi için uygun bir ortam sunmaktadır.

Bir paketin içerisinde diğer paketlerle oluşacak bir isim çakışmasını düşünmeden sınıf tanımı yapılabilmektedir.

Paketler hiyerarşik bir yapıda kullanılabilirler ve bir paketin kullanılabilmesi için kullanılacağı sınıfın içerisine dahil edilmelidirler.

PAKETLER

- package
- paketin içerisindeki sınıflar
- isimsiz paket

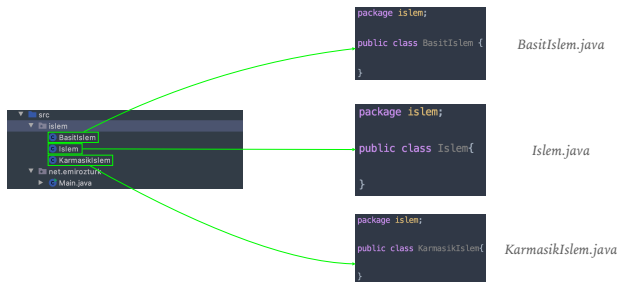
Islem.java

```
package islem;  
  
public class Islem{  
  
}
```

Bir paketin tanımlanabilmesi için java kaynak dosyasının başına "package paket ismi" yazılması gerekir.

Paket isminin yazılmasından sonra tanımlanan tüm sınıflar bu pakete dahil olur. Paket ismi verilmediği takdirde yazılan sınıflar varsayılan isimsiz bir paket altında toplanır.

PAKETLER



Paket altındaki tüm sınıfların aynı dosyada olması gerekmemektedir.

Kaynak dosyaların bulunduğu klasörün altında paket açıldıktan sonra ayrı ayrı tüm sınıflar paket klasörü içerisinde tanımlanabilmektedir. Bu sınıflar için oluşturulmuş kaynak dosyaların başında dahil oldukları paketin yazılması yeterlidir.

PAKETLER

```
package islem;  
  
public class Islem{  
  
}  
  
class YardimciIslem{  
  
}
```

```
package islem;  
public class Islem{  
}  
public class YardimciIslem{  
}
```

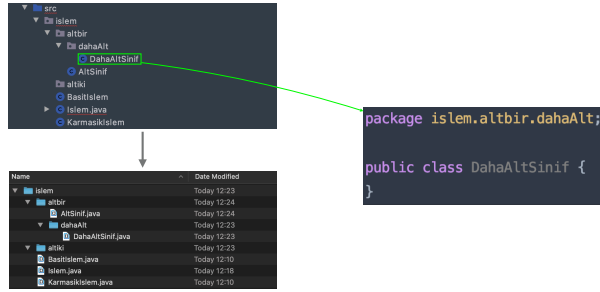
Class "YardimciIslem" is public, should be declared in a file named "YardimciIslem.java"
Make "YardimciIslem" not public: View More actions

Bir kaynak dosya içerisinde birden fazla sınıf da tanımlanabilmektedir.

Bu sınıflar ayrıca dışarıdan erişilemeyecek bir biçimde default veya private olarak tanımlanabilmektedirler.

Dışarıdan erişilebilecek bir public sınıf aynı paket içerisinde tanımlandığında sınıf ismine ait başka bir dosya içerisinde tanımlanması istenmektedir.

PAKETLER



Paketler genellikle hiyerarşik bir biçimde saklanmaktadır.

Dosya sisteminde bu hiyerarşi iç içe klasörlerde saklama yolu ile yapılırken, isimlendirmeler her alt paket noktadan sonra gelecek şekilde yapılır.

Örneğin islem paketi altında altbir paketi varsa bu islem.altbir şeklinde isimlendirilir.

Bir alt paket daha olması durumunda nokta koyularak devam edilir.

PAKETLER / ERİŞİM DÜZEYLERİ

	private	default	protected	public
Sınıf içi	✓	✓	✓	✓
Aynı paket alt sınıflar		✓	✓	✓
Aynı paket diğer sınıflar		✓	✓	✓
Farklı paket alt sınıflar			✓	✓
Farklı paket diğer sınıflar				✓

private ile yalnızca sınıf içinden erişim sağlanırken, default ile paket içerisinde erişim sağlanabilmektedir. protected, default'tan farklı olarak başka paketlerdeki alt sınıflara da erişimi açmakta, public ise her yerden erişim sağlamak için kullanılmaktadır.

PAKETLER

- > import PaketAdı.AltPaket.SınıfAdı
- > import PaketAdı.*
- > import PaketAdı.AltPaket.*


Bir paketin içerisindeki sınıfların başka bir paket içerisinden kullanılabilmesi için import anahtar kelimesi kullanılır.

Kullanılacak sınıfların ismi paket adı ile birlikte hiyerarşik olarak tam yol halinde verilmelidir.

Eğer bir paketteki tüm sınıflar kullanılacaksa paket adından sonra * ifadesi kullanılabilir.

* kullanıldığında bir paketin alt paketleri import edilmez. Bu sebeple alt paketlerin ayrıca import edilmesi gerekmektedir.

PAKETLER



```
package net.emirozturk;

public class Main {

    public static void main(String[] args) {
        Ismem i = new Ismem();
    }
}

Cannot resolve symbol 'Ismem'
Import class

package net.emirozturk;

import Ismem;

public class Main {

    public static void main(String[] args) {
        Ismem i = new Ismem();
        Ismem i1 = new Ismem();
        Ismem i2 = new Ismem();
    }
}
```



```
package net.emirozturk;

import islem.Islem;

public class Main {

    public static void main(String[] args) {

        Islem i = new Islem();
        BasitIslem b1 = new BasitIslem();
        GelişmişIslem g1 = new GelişmişIslem();
    }

}
```



Diğer paketlerdeki sınıfları kullanmak için sınıf ismi import ile yazılır (import işlem.Islem).

PAKETLER

```
package net.emrozturk;

import islem.Islem;

public class Main {

    public static void main(String[] args) {
        Islem i = new Islem();
        islemisim bi = new islemisim();
        islemisim das = new islemisim();
    }
}
```



```
package net.emrozturk;

import islem.BasitIslem;
import islem.Islem;

public class Main {

    public static void main(String[] args) {
        Islem i = new Islem();
        BasitIslem b1 = new BasitIslem();
        BasitIslem das = new BasitIslem()
    }
}
```



islem paketi içerisindeki Islem sınıfı alınsa da BasitIslem sınıfında hala erişilememektedir. Bu sebeple islem paketi içerisinde BasitIslem sınıfı da import edilmelidir.

PAKETLER

```
package net.emirozturk;

import islem.BasitIslem;
import islem.Islem;

public class Main {

    public static void main(String[] args) {

        Islem i = new Islem();
        BasitIslem b1 = new BasitIslem();
        BasitIslem das = new BasitIslem();

    }

}
```



```
package net.emirozturk;

import Islem.*;

public class Main {

    public static void main(String[] args) {
        Islem i = new Islem();
        BasitIslem bi = new BasitIslem();
        BasitIslem bas = new BasitIslem()
    }
}
```



islem paketi içerisindeki tüm sınıfların import edilebilmesi için * kullanılabilir. Fakat islem paketinin alt paketleri * ile dahil edilmediği için DahaAltSinif hala tanınmamaktadır. Bu sebeple bu alt paketin de ayrıca import edilmesi gerekmektedir.

PAKETLER

```
package net.emirozturk;

import islem.*;

public class Main {

    public static void main(String[] args) {
        Islem i = new Islem();
        BasitIslem bi = new BasitIslem();
        DahaAltSinif das = new DahaAltSinif();
    }
}
```



```
package net.emirozturk;

import islem.*;
import islem.altbir.dahaAlt.DahaAltSinif;

public class Main {

    public static void main(String[] args) {
        Islem i = new Islem();
        BasitIslem bi = new BasitIslem();
        DahaAltSinif das = new DahaAltSinif();
    }
}
```

```
net.emirozturk
├── islem
│   ├── Islem
│   ├── IslemAlt
│   └── IslemAltSinif
│       ├── DahaAltSinif
│       ├── AltSinif
│       ├── Islem
│       ├── BasitIslem
│       ├── Islem.java
│       └── KamusalIslem
```

PAKETLER

```
package net.emirozturk;

import islem.altbir.dahaAlt.DahaAltSinif;

public class Main {

    public static void main(String[] args) {
        var das = new DahaAltSinif();
    }
}
```



```
public class Main {

    public static void main(String[] args) {
        var das = new islem.altbir.dahaAlt.DahaAltSinif();
    }
}
```

```
net.emirozturk
├── islem
│   ├── Islem
│   ├── IslemAlt
│   └── IslemAltSinif
│       ├── DahaAltSinif
│       ├── AltSinif
│       ├── Islem
│       ├── BasitIslem
│       ├── Islem.java
│       └── KamusalIslem
```

Paket dahil etmek istenilmediği durumlarda açık isim verilerek de sınıflar kullanılabilirler.