NESNEYE YÖNELİK PROGRAMLAMA

Soyut Sınıflar, Arayüzler ve Paketler

Emir Öztürk

SOYUT SINIFLAR

- Sınıf tanımıyla aynı
- abstract anahtar kelimesi
- Örnek oluşturamama

Soyut sınıf tanımı sınıf tanımıyla aynı şekilde gerçekleştirilir.
Farklı olarak soyut sınıfın tanımlanmasında abstract anahtar kelimesi kullanılmaktadır.
Soyut sınıflardan örnek oluşturulamamaktadır.

SOYUT SINIFLAR

- Ortak özelliklerin toplanması
- Nesne oluşturamayacak genellikte olması

Belirli sınıflar ortak özellikleri toplayacak kadar genel olabilir fakat bu sınıflardan nesne oluşturulması anlamsız olabilmektedir. Sınıf hiyerarşisinde sınıfların genelleştirilmesi ile bu sınıflardan nesne oluşturulamama seviyesine ulaşılabileceğinden böyle durumlarda en uygun olan soyut sınıfların kullanımıdır.

SOYUT SINIFLAR – ARAYÜZLER

- Soyut Sınıf
  - Çok sınıf, ortaklık
  - Ortak metot ve alanlar, erişim düzeyleri
  - Non static, non final

Birbirlerine çok yakın sınıflar arası kod paylaşımı,
Soyut sınıfı kullanan diğer sınıflarda ortak olan metot ve alanların çokluğu / public dışındaki erişim belirleyicilerine sahip metotların olması,
Static veya final olmayan alanların olması durumlarında soyut sınıfların kullanılması tercih edilmektedir.

SOYUT SINIFLAR – ARAYÜZLER

- Arayüz
  - Çok sınıf, bağımsızlık
  - İmplementasyon bağımsızlığı
  - Çoklu kalıtım

Birbirleri ile alakalı olmayan fazla sayıda sınıf ortak özellikleri kullanacaksa (örneğin comparable gibi bir arayüz bir çok sınıf tarafından kullanılmaktadır.),
Türler belirli fakat implementasyon bağımsızsa,
Çoklu kalıtım ihtiyacı varsa arayüzlerin kullanılması tercih edilmektedir.

SOYUT SINIFLAR

```
class A{

}
class B extends A{

}
public class Main {

    public static void main(String[] args) {
        A a = new A();
        B b = new B();
    }
}
```

```
abstract class A{

}
class B extends A{

}
public class Main {

    public static void main(String[] args) {
        A a = new A();
        B b = ne
    }
}
```

'A' is abstract; cannot be instantiated

Make 'A' not abstract ⌥⏎ More actions... ⌥⏎

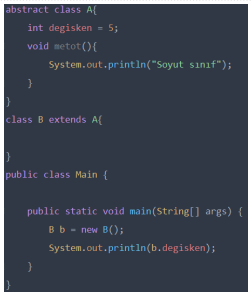
Soyut sınıflar tanımlanırken abstract kelmesi kullanılmaktadır.
Abstract olarak tanımlanmış bir sınıfın herhangi bir yerde örneği alınamaz. Bu sınıf yalnızca diğer sınıflardan türetilmek için kullanılır.

SOYUT SINIFLAR

- Özelliklere sahip olabilme
- Özelliklerinin değerinin atanabilmesi
- Özelliklere erişim

Soyut sınıflardan bir örnek alınamasa da soyut sınıflarda özellikler tanımlanabilmektedir.
Bu özelliklere varsayılan bir değer atanabilmesi de mümkündür.
Bu sınıfı kalıtan başka bir sınıf tanımlandığında bu özelliğe erişim sağlanabilmektedir.

SOYUT SINIFLAR



```
abstract class A{
    int degisken = 5;
    void metot(){
        System.out.println("Soyut sinif");
    }
}
class B extends A{

}
public class Main {

    public static void main(String[] args) {
        B b = new B();
        System.out.println(b.degisken);
    }
}
```

Örnekte görüldüğü gibi bir A soyut sınıfında degisken isimli 5 değerine sahip bir değişken tanımlanmıştır. B sınıfı bu sınıfı kalıttığında bu değere erişim sağlanabilmektedir. B sınıfından bir nesne oluşturulduğunda default olarak tanımlanan bu değişken ekrana 5 değerini gösterecektir.

SOYUT SINIFLAR

- Metotlara sahip olabilme
- Metotların gövdesinin tanımlanabilmesi
- Metotlara erişim

Soyut sınıflarda metotlar da tanımlanabilmektedir. Bu metotların gövdeleri yazılarak diğer sınıflar tarafından da kullanılabilmektedir. Soyut bir sınıftan kalıtılan başka bir sınıf tanımlanmış bir metodu çiğnemiyorsa (override) soyut sınıfta tanımlanmış metot implementasyonu kullanılacaktır.

SOYUT SINIFLAR

```java
abstract class A{
    void metot(){
        System.out.println("Soyut sınıf");
    }
}
class B extends A{

}
public class Main {

    public static void main(String[] args) {
        B b = new B();
        b.metot();
    }
}
```

Soyut sınıf

```java
abstract class A{
    void metot(){
        System.out.println("Soyut sınıf");
    }
}
class B extends A{
    void metot(){
        System.out.println("Aşırı yüklenmiş metot");
    }
}
public class Main {

    public static void main(String[] args) {
        B b = new B();
        b.metot();
    }
}
```

Çiğnenmiş Metot

Soyut sınıflar diğer sınıflarda olduğu gibi metotlar ve özellikler içerebilmektedirler. Aynı zamanda final kelimesi kullanılmadığı sürece bu metotlar override edilebilmektedirler.
Override edilmediği sürece metotlar normal sınıflarda olduğu gibi kalıtım sırasında bulunabilen en yakın sınıftan çağırılırlar.

SOYUT SINIFLAR

- Soyut metotlar
- Soyut metot olan sınıfın soyut sınıf olması
- abstract
- Metot gövdesi

Soyut sınıflarda soyut metotlar da tanımlanabilmektedir. Bu metotlar bir gövde içermemekte ve override edilmesi gerekmektedir.
Soyut metotlar yalnızca soyut sınıf içerisinde tanımlanabilirler.
Soyut bir metot yalnızca sınıfın içerisinde bu metodun bulunacağını ve bu metodun nasıl bir imzaya sahip olacağını belirler. İmplementasyon detayı kalıtımı gerçekleştiren sınıfa ait olacaktır.

SOYUT SINIFLAR

```java
abstract class A{
    abstract void metot(){
    }
}
```

Abstract methods cannot have a body

Delete method body ⌥⇧↩ More actions... ⌥↩

```java
abstract class A{
    abstract void metot();
}
class B extends A{
    void metot(){
        System.out.println("Aşırı yüklenmiş metot");
    }
}
public class Main {

    public static void main(String[] args) {
        B b = new B();
        b.metot();
    }
}
```

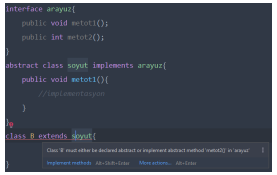
Soyut tanımlanan bir metot gövde içermemektedir.
Bu metodun kullanılabilmesi için diğer sınıflar içerisinde override edilmesi gerekmektedir.

SOYUT SINIFLAR

```
interface arayuz{
    public void metot1();
    public int metot2();
}
abstract class soyut implements arayuz{
    public void metot1(){
        //implementasyon
    }
    public int metot2(){
        //implementasyon
        return 0;
    }
}
class B extends soyut{
    //metot1 ve metot2'nin tanımlanmasına gerek yoktur.
}
```

Soyut sınıflar aynı zamanda arayüzleri de içerebilirler. Soyut sınıfların bu arayüzleri içermesi durumunda arayüzün tanımlanması gereken metotlarının tamamı soyut sınıf üzerinde tanımlanabilir.
Böylece soyut sınıftan türetilmiş başka bir sınıf arayüzün metotlarını tekrar tanımlamadan kullanılabilir. B sınıfından oluşturulan bir nesne metot1 veya metot2 metotlarını çağırdığında bu metotlar temel sınıfta tanımlı oldukları için sorunsuz bir şekilde ulaşabileceklerdir.

SOYUT SINIFLAR



```
interface arayuz{
    public void metot1();
    public int metot2();
}
abstract class soyut implements arayuz{
    public void metot1(){
        //implementasyon
    }
}
class B extends soyut{
    public int metot2() {
        return 0;
    }
}
```

Soyut sınıflar arayüzlerin tüm metotlarını içermek zorunda değildirler. Herhangi bir soyut sınıfta eksik bir tanım yapıldığında kod hata vermeyecektir. Fakat bu soyut sınıftan kalıtılmış başka bir sınıf olması durumunda bu sınıfta eksik kalan metotların tamamlanması gerekmektedir. Örneğin arayuz interface'ini içeren bir soyut isimli abstract class, arayüzün yalnızca metot1() isimli metodunu tanımlamıştır. Bu durumda derleyici hata vermemektedir. Soyut sınıftan türetilmiş bir B sınıfı ise tanımlandığı anda metot2() metodunun eksik olduğu hatasını verecektir. Bu durumda kalan metotları (metot2()) bu sınıf içerisinde tanımlamak gerekmektedir.

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 içeremeyecek 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

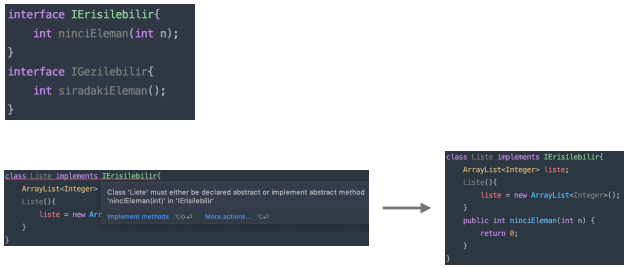
İlk değeri verilmemiş değişkenler hata verdiği için arayüzlerde ilk değişken tanımı gereklidir.

ARAYÜZLER

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

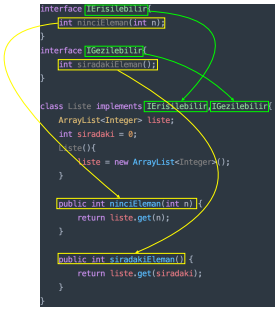
Yazılan bir arayüzü bir sınıfın kullanabilmesi için implements anahtar kelimesi kullanılır.
Bir sınıf, arayüz metotlarının tamamını içermelidir fakat sınıf sadece arayüz metotlarını içermek zorunda değildir.
Sınıfta arayüz metotlarından farklı olarak kendine ait metotlar da bulunabilir.

ARAYÜZLER



IErisilebilir arayüzünü içeren bir Liste sınfında 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



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 koyulmaktadır.

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.emirozturk.Liste' More actions...

Bir sınıfın tanımı miras alınan bir üst sınıf ile yapılabildiği gibi arayüzlerle de yapılabilmektedir.
Metot çağırımı sırasında bu arayüzler çalışma anında metodu çağıran 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 çağırılabilmektedirler.

C++ VE ARAYÜZLER

- C++'ta arayüz anahtar kelimesi bulunmamaktadır
- Bunun yerine soyut sınıflar ile aynı mekanizma oluşturulur

```cpp
#include <iostream>
#include <vector>
#include <stdexcept>

using namespace std;

class IErisilebilir {
public:
    virtual int ninciEleman(int n) const = 0;
    virtual ~IErisilebilir() {}
};
class Liste : public IErisilebilir {
private:
    vector<int> liste;

public:
    Liste() {}

    int ninciEleman(int n) const override {
        if(n >= 0 && n < liste.size())
            return liste[n];
        else
            throw out_of_range("Gecersiz indeks: " + to_string(n));
    }

    void elemanEkle(int x) {
        liste.push_back(x);
    }
};
```

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.

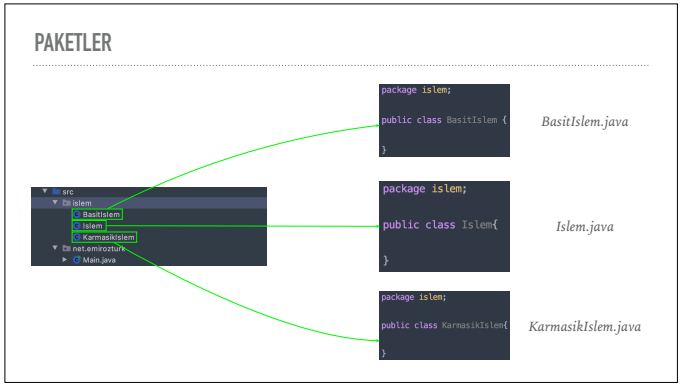
PAKETLER

- package
- paketin içerisindeki sınıflar
- isimsiz paket

Islem.java

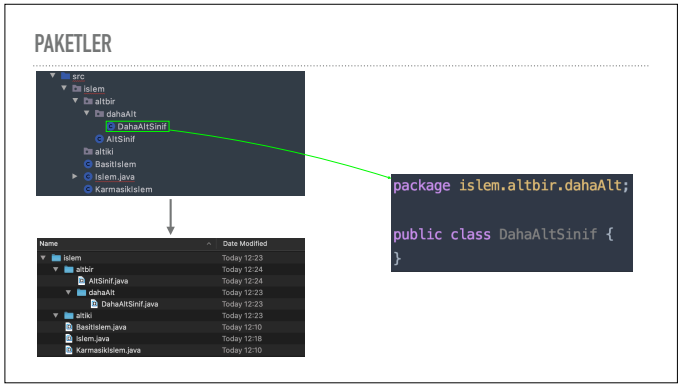
```java
package islem;

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.



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.

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 genellikle hiyerarşik bir biçimde saklanmaktadırlar.
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 DÜZEYLERI

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

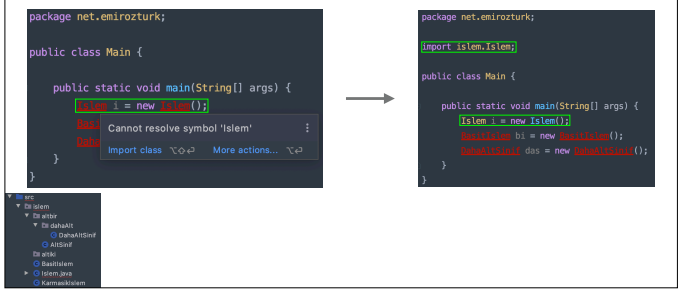
private ile yalnızca sınıf içinden erişim sağlanırken, default ile paket içerisinde erişim sağlanabilmektedir. protected, default'tan farklı olarak başka paketlerdeki alt sınıflara da erişimi açmakta, public ise her yerden erişim sağlamak için kullanılmaktadır.

PAKETLER

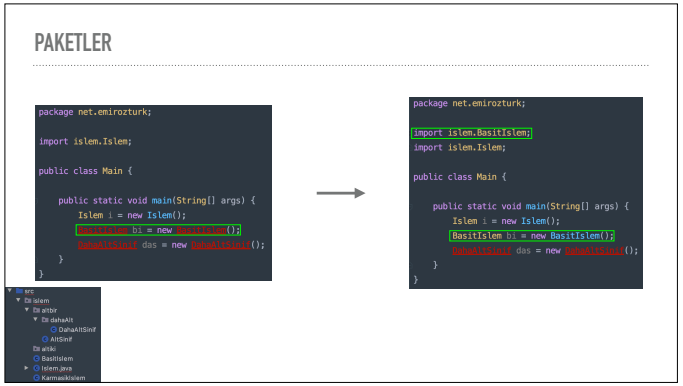
- import PaketAdı.AltPaket.SınıfAdı
- import PaketAdı.*
- import PaketAdı.AltPaket.*

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

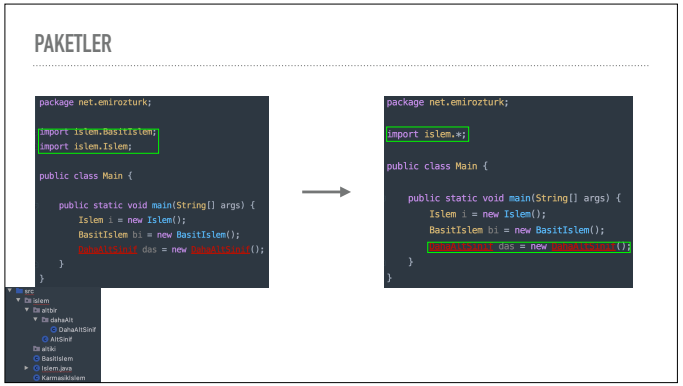
PAKETLER



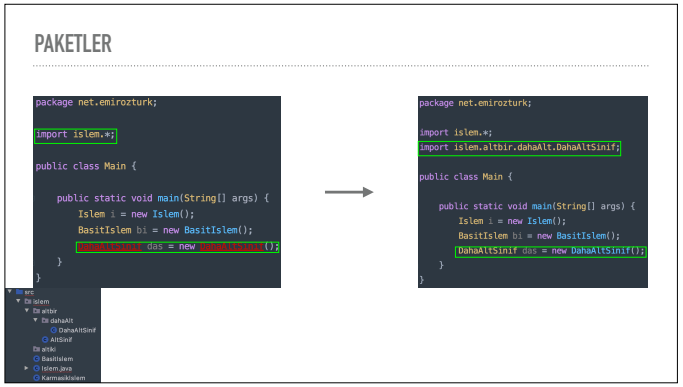
Diğer paketlerdeki sınıfları kullanmak için sınıf ismi import ile yazılır (import islem.Islem).



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



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

```java
package net.emirozturk;

import islem.altbir.dahaAlt.DahaAltSinif;

public class Main {

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

```java
public class Main {

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

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