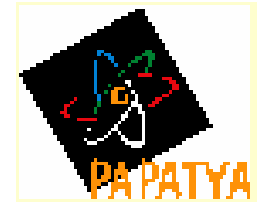
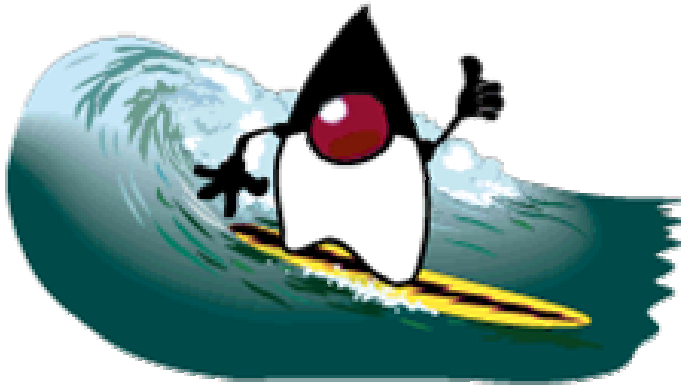


Hata Ayıklamanın Ötesi...

(Assertion)



Assertion

- Assertion kelimesinin Türkçe karşılığı iddia, birşeylerin doğruluğunu ispat etmek anlamlarına gelir.
- Assertion özelliği, J2SE 1.4 versiyonu ile birlikte gelen yeni bir özelliktir.
- Bu yeni gelen özellik sayesinde hata ayıklama (*debugging*) ve yazılan kodların doğruluğunu ispat etme süreçleri çok daha basite indirgenmektedir.

Hata Ayıklama (*Debugging*) - I

- Hata ayıklamak (*debugging*) ne demek ?
- Hata ayıklama işlemi, hatanın algılanmasından sonra gelen bir süreçtir ve süreci uygulamak için bir çok yöntem bulunur.
- En bilindik yöntemlerden biri hatalı olduğuna inanılan kod yığınlarının arasına **`System.out.println()`** komutları serpiştirilerek uygulamanın akışı takip edilmeye çalışılır.

Hata Ayıklama (Debugging) - II

```
public double ortalamaBul(double[] dizi) throws Exception{
    System.out.println("ortalamaBul metodunun icinde.."); //dikkat
    if (dizi == null ) {
        System.out.println("dizi null gelmis, hata firlatilacak"); //dikkat
        throw new Exception("Gonderilen dizi null");
    } else if ( dizi.length == 0) {
        System.out.println("dizi icerisinde eleman yok"); //dikkat
        return 0 ;
    }

    double ortalama = 0 ;
    for (int i=0; i<dizi.length; i++){
        System.out.println("dizi["+i+"] : " + dizi[i]); //dikkat
        ortalama += dizi[i] ;
    }

    System.out.println("1- ortalama : " + ortalama); //dikkat
    ortalama = ortalama / dizi.length ;

    System.out.println("2- ortalama : " + ortalama); //dikkat
    return ortalama ;
}
```

- Diğer yöntem ise "*Java Platform Debugger Architecture*" mimarisini kendi içerisine entegre etmiş bir editör ile çalışmaktır.
 - Eclipse
 - VisualSlickEdit
 - JBuilder
 - CodeGuide
 - gibi...

Assertion özelliğini kullanmak

- Assertion özelliğini yazılan kodların içerisine yerleştirmek çok kolaydır.
- Assertion, koşullar gerçekleşmediği zaman hata fırlatan bir mekanizmadır.
- Assertion özelliğini kullanmanın iki yolu vardır.

Assertion özelliğini kullanmak – Birinci Yol

- Birinci yol sadece basit bir ifadeden oluşur.

```
assert ifade ;
```

Yukarıda belirtilen ifade **true** ise sorun çıkmaz ama şayet bu ifade **false** ise sorun var demektir ve hata (AssertionError) fırlatılır.

Örnek - Birinci Yol

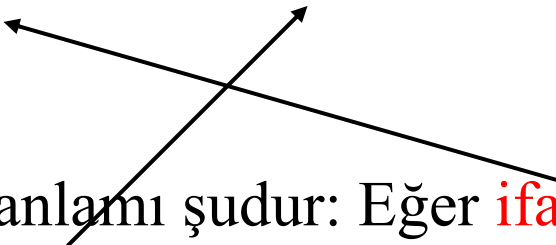
```
public class OrnekSinif {  
    public void ornekMetod( int parametre ) {  
        Asinifi a = null;  
        // ... Asinifi tipinde bir nesne alınmaya calisiliyor  
        // Islem basarili mi basarisiz mi olmus ?  
        assert a != null; ← ★  
    }  
}
```

Yukarıdaki assert ifadesinin kullanılmasındaki amaç, Asinifi sınıfı tipindeki **a** referasının acaba Asinifi sınıfına ait bir nesneye mi bağlandığını kontrol etmektir.

Assertion özelliğini kullanmak – İkinci Yol

```
assert ifade_1 : ifade_2;
```

Yukarıdaki ifadenin anlamı şudur: Eğer **ifade_1** false değeri geri dönerse, **ifade_2** deki değeri hata olarak fırlat.



Örnek - İkinci Yol

```
public class OrnekSinif {  
    public void ornekMetod( int parametre ) {  
        Asinifi a = null;  
        //.. Asinifi tipinde bir nesne almaya calisiliyor  
        // Islem basarili mi basarisiz mi olmus ?  
        assert a != null : "assert a != null : Nesne elde edilemedi, parametre=" + "parametre" ;  
    }  
}
```

- Bu gösterimimizde **a** referansı eğer Asinifi tipinde bir nesneye bağlanmamış ise yeni bir AssertionError tipinde bir hata fırlatılacaktır.
- Yanlız buradaki fark, bu AssertionError sınıfının yapılandırıcısına bizim bazı bilgiler gönderiyor olmamızdır.

Assertion ve derleme (*compile*) - I

- Assertion özelliği Java programlama diline yeni bir anahtar kelime kazandırmıştır.
- Java programlama dili daha evvelden yazılmış diğer uygulamalar için bir tehlike oluşturabilir.
- Bu tehlike geriye doğru uyumluluğun kalkması (*backwards compatibility*) yönündedir.

Assertion ve derleme (*compile*) - II

```
public class Uyum {  
    public static void main(String args[]) {  
        String assert = args[0] ; // dikkat  
        for (int i=0; i<10; i++) {  
            System.out.println( i + ":" + assert);  
        }  
    }  
}
```

- Yukarıdaki örneğimiz henüz assertion özelliği ortalarda yokken yazılmış olsun.
- Bu uygulamamızda, kullanıcıdan gelen ilk değeri String tipinde olan ve **assert** isimli bir referansa bağlanmaktadır.
- Yazılan Java kodlarının içerisinde assert anahtar kelimesi referans adı olarak geçiyorsa ve assertion özelliğini kullanmak istemiyorsanız, kısacası ben eski usül çalışmak istiyorum diyorsanız bazı ayrıntılara dikkat etmeniz gerekir.

Assertion ve derleme (*compile*) - III

```
public class Uyum {  
    public static void main(String args[]) {  
        String assert = args[0] ; // dikkat  
        for (int i=0; i<10; i++) {  
            System.out.println( i + ":" + assert);  
        }  
    }  
}
```

Assertion özelliğini kullanmamak için...

```
> javac -source 1.3
```

Assertion özelliğini kullanmak için...

```
> javac -source 1.4
```

Assertion özelliğini nasıl kontrol ederim ? - I

- Assertion özelliğinin kıymetli kılan en önemli faktör, bu özelliğin çalışma esnasında kapatılıp açılabiliyor olmasıdır.
- Örneğin bir uygulamanın geliştirilmesi esnasında assertion özelliği açık tutulabilir.
- Tahmin edilebileceği üzere assertion özelliğin açık tutulması belli bir performans kaybına sebebiyet verecektir.
- Fakat uygulamanın gelişimi tamamlandığı zaman assertion özelliği çalışma anında kapatılarak (biraz sonra gösterilecek) bu performans kaybı engellenmiş olur.

Assertion özelliğini nasıl kontrol ederim ? - II

```
public class AssertTestBir {  
    public static void main(String[] args) {  
        assert 1 == 10 ;  
    }  
}
```

- *AssertTestBir.java* uygulamasını *javac -source 1.4* komutu ile derledikten sonra aşağıdaki gibi çalıştırılırsa...

```
> java AssertTestBir
```



```
> java -ea AssertTestBir
```

Assertion özelliğini nasıl kontrol ederim ? - III

```
public class AssertTestIki {  
    public static void main(String[] args) {  
        int gelDeger = ciftSayiAl(Integer.parseInt(args[0]));  
        assert gelDeger % 2 == 0; // dikkat  
    }  
  
    public static int ciftSayiAl(int sayi) {  
        return sayi * 3;  
    }  
}
```

Önce derleme (*compile*) aşaması

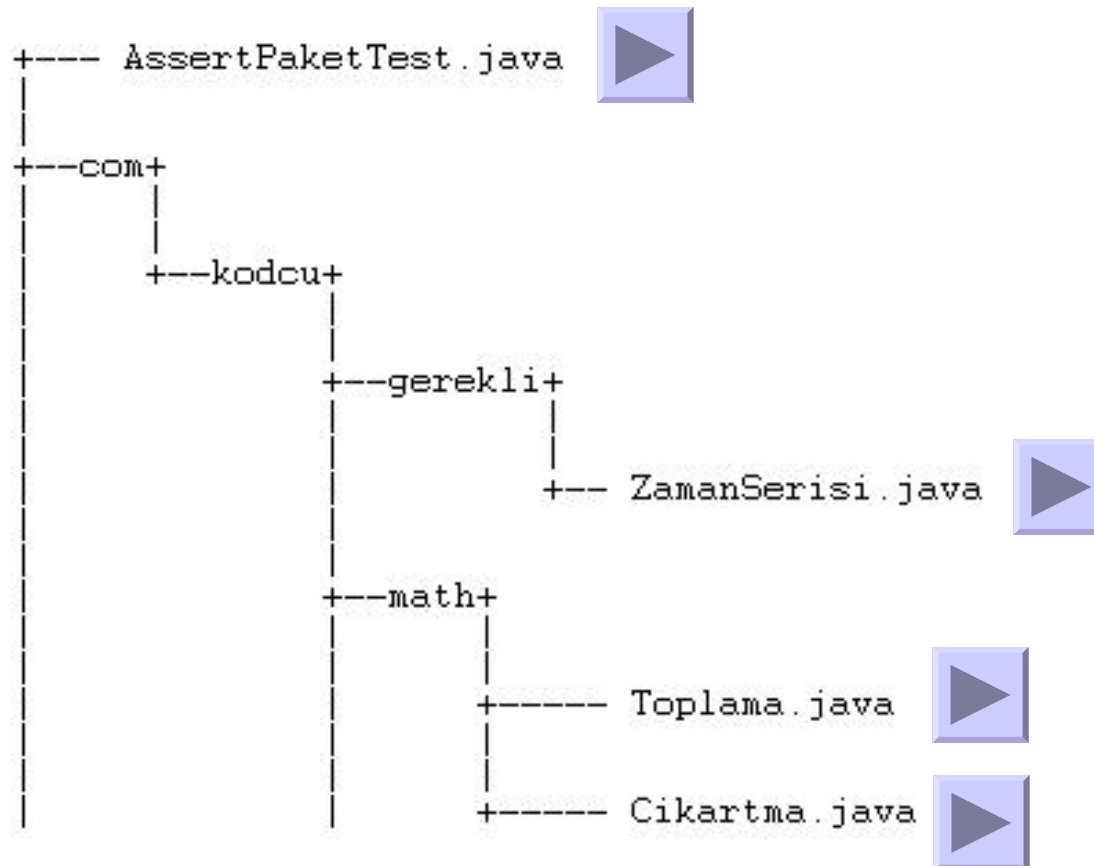
> javac -source 1.4 AssertTestIki.java

Sonra çalıştırma (*run*) aşaması

> javac -ea AssertTestIki 5

Paket kontrolleri

- Şimdi aşağıdaki gibi bir yapımız olsun.



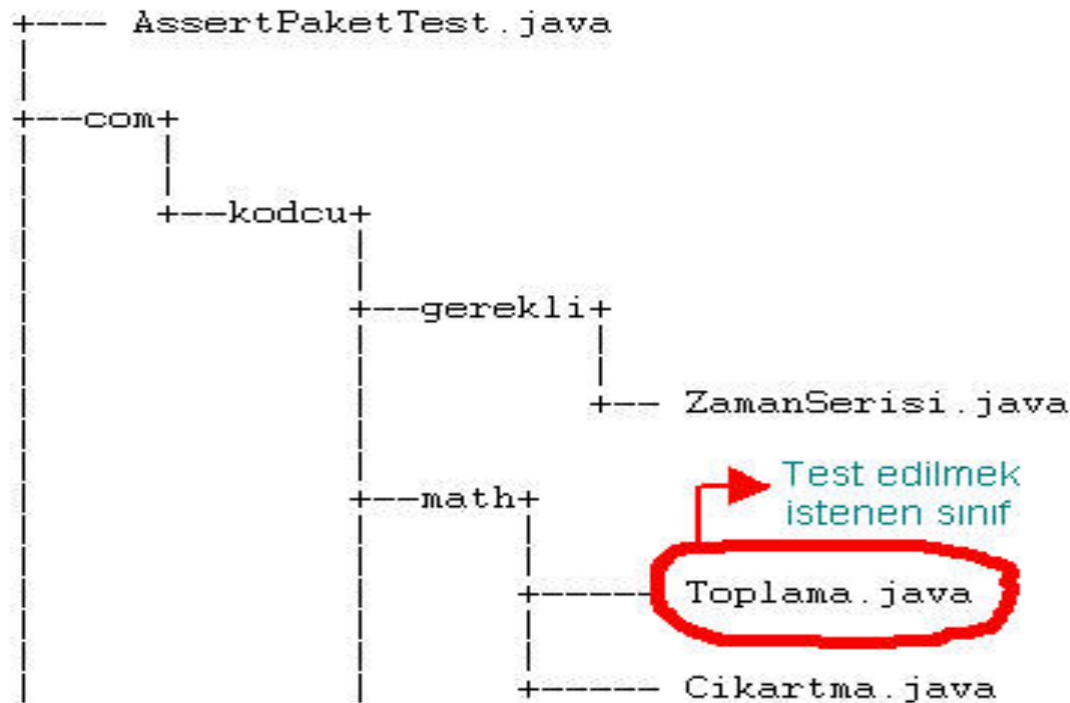
Tüm paketler için assertion özelliği açık

- *AssertPaketTest* sınıfının içerisindeki tüm paket ve bunlar içerisindeki sınıflar için assertion özelliğinin açık (etkin) olması isteniyorsa aşağıdaki komutun yazılması yeterli olacaktır.

```
> java -ea AssertPaketTest
```

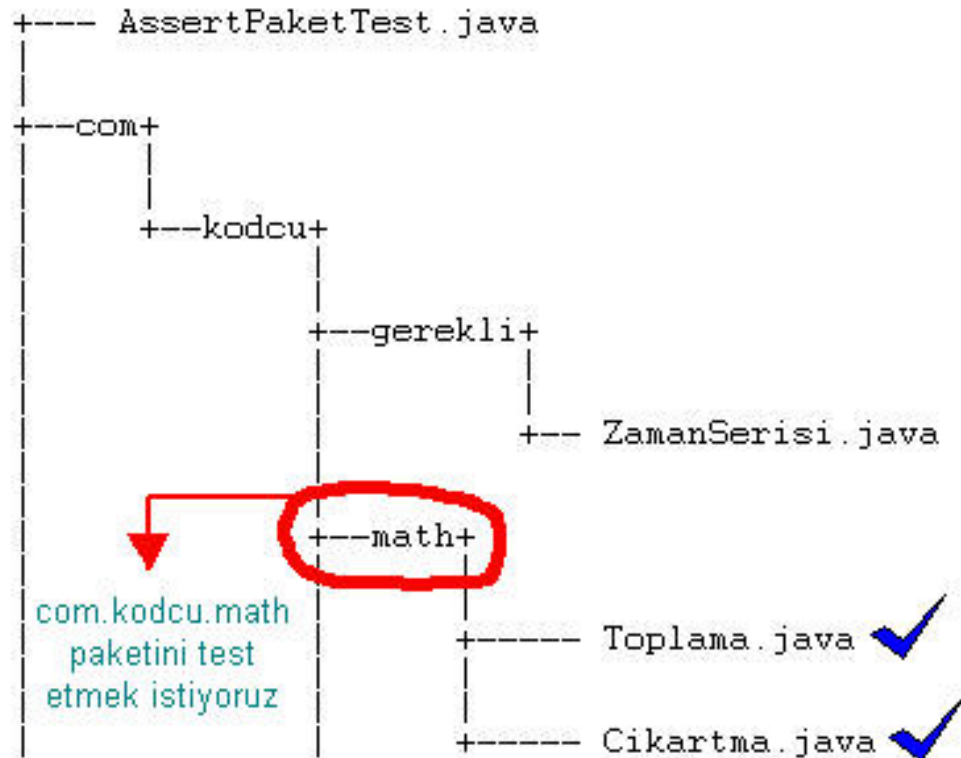
Toplama işlemlerinde bir hata var sanki....

- Assertion özelliğinin sadece *com.kodcu.math.Toplama* sınıfı için açılmak istenirse.



> `java -ea:com.kodcu.math.Toplama AssertPaketTest`

Sadece *com.kodcu.math.** altındaki tüm sınıflar için
assertion özelliği açmak istersek....



- > `java -ea:com.kodcu.math AssertPaketTest` ❌
- > `java -ea:com.kodcu.math... AssertPaketTest`

Assertion özelliği açık mı ? Kapalı mı ?

```
package com.kodcu.gerekli;

public class ZamanSerisi {

    public ZamanSerisi() {

        boolean assertionEtkinMi = false;

        assert(assertionEtkinMi = true) ;

        if (assertionEtkinMi) {
            System.out.println("com.kodcu.gerekli.ZamanSerisi sinifi"+
                               " için Assertion etkin");
        }
    }
}
```

Yukarıdaki sınıfımızın yapılandırıcısındaki assert anahtar kelimesinin olduğu satıra yakından bakacak olursak, burada bir karşılaştırma değil bir atama olduğunu görürüz.

AssertionError istisnalarını yakalamak

```
public class AssertionHataYakalama {  
    public static void main(String args[]) {  
        try {  
            int i = 10 ;  
            assert i == 20 ; // kesin assertion hatasi firlatilacak  
        } catch (AssertionError ae) {  
            StackTraceElement ste[] = ae.getStackTrace();  
  
            if (ste.length>0) {  
                StackTraceElement sonuc = ste[0];  
                System.out.println( "Assertion hatasi : "+  
                    sonuc.getFileName()+" \n satir :" + sonuc.getLineNumber() );  
            } else {  
                System.out.println( "Herhangi bir bilgi yok" );  
            }  
  
            throw ae; // cok onemli  
        }  
    }  
}
```

Kural : Komut satırından girilmiş olan verilerin kontrolü için assertion özelliği kullanılmamalıdır.

- Assertion özelliği, uygulamanın kendi içerisinde tutarlılığını sağlamak için kullanılmalıdır; kullanıcının uygulama ile olan tutarlılığını sağlamak için değil.

```
public class KuralBirHatali {  
    static public void main( String args[] ) {  
  
        assert args.length == 3; // Yanlış kullanım  
        double x = Double.parseDouble( args[0] );  
        double y = Double.parseDouble( args[1] );  
        double z = Double.parseDouble( args[2] );  
    }  
}
```

Kural : Assertion özelliği, if (koşul)..... yerine kullanılmamalıdır.

- Kritik nokta, çalışma anında assertion özelliğinin kapatılması ile göz ardı edilir.

```
public class KuralIkiHatali {  
    private int port;  
    public void dinle() {  
        // Yanlis Kullanim  
        assert port > 1024 : "Bu port numarasi kullanilamaz "+ port;  
        // ...  
    }  
}
```


Kural : **public** erişim belirliyecisine sahip olan yordamlara gönderilen parametreleri düz şekilde kontrol etmek amacıyla **assert** özelliği kullanılmamalıdır.

```
public class KuralUcHatali {  
    public double varyansHesapla( double[] dizi ) {  
        assert dizi != null ; // Yanlis Kullanim  
        // ..  
    }  
    //..  
}
```

Kural : Kullanıcıdan gelen verilerin mantık çerçevesinde olup olmadığı assertion özelliği ile kontrol edilmemelidir.

```
public void telefonNumarasiniAl( String telefonNumarasi ) {  
    if ( telefonNumarasi.length() == 7 ) {  
        // ...  
    } else if ( telefonNumarasi.length() == 11 ) {  
        // ...  
    } else {  
        // Yanlis Kullanim  
        assert false : "telefon numarasi 7 veya 11 haneden olusmali";  
    }  
}
```

Kural : Uygulamanın genel akışında assertion özelliğinin bir rolü olmamalıdır.

```
public class Test {  
    static private boolean hatayiSakla(Exception ex) {  
        // Hatayi saklama islemi  
        // ...  
        return true;  
    }  
    static public void main( String args[] ) throws Exception {  
        // ...  
        try {  
            // ...  
        } catch (Exception ex) {  
            // Yanlis Kullanim  
            assert hatayiSakla(ex);  
        }  
    }  
}
```

Kural : `private` erişim belirleyicisine sahip olan yordamlara gönderilen parametrelerin kontrolünde `assertion` özelliği kullanılabilir.

- **`private`** erişim belirleyicisine sahip olan yordamlar dışarıdan ulaşılamaz.
- Bu tip yordamlar işlerin esas yapıldığı ve yanlış parametre gelmesinin affedilemeyeceği yerlerdir.

```
private double standartSapmayiBul( double[] dizi ) {  
    assert dizi != null ;  
}
```

Kural : Olmaz ise olmaz durumlarını yakalamak için assertion özelliği kullanılabilir.

```
private void hazirla() throws IOException {  
    Properties p = new Properties() ;  
    BufferedInputStream okuyucu = new BufferedInputStream(  
        new FileInputStream("c:\\gerekli\\bilgiler\\klc.properties"));  
  
    p.load(okuyucu);  
  
    kullanıcıAdi = (String) p.get("KULLANACIADI");  
    sifre = (String) p.get("SIFRE");  
  
    assert kullanıcıAdi != null ; // dogru bir yaklasim  
    assert sifre != null ; // dogru bir yaklasim  
}
```

Sorular ...

