Java'da
Program Denetimi
ve
Operatörler

Atamalar

```
int a ;
a=4 ;   // doğru bir atama
4=a ;   // yanlış bir atama!
```

Temel (Primitive) Tiplerde Atama

```
int a, b ;
a=4 ;
b=5 ;
a=b ;
```

Sonuç : a=5, b=5

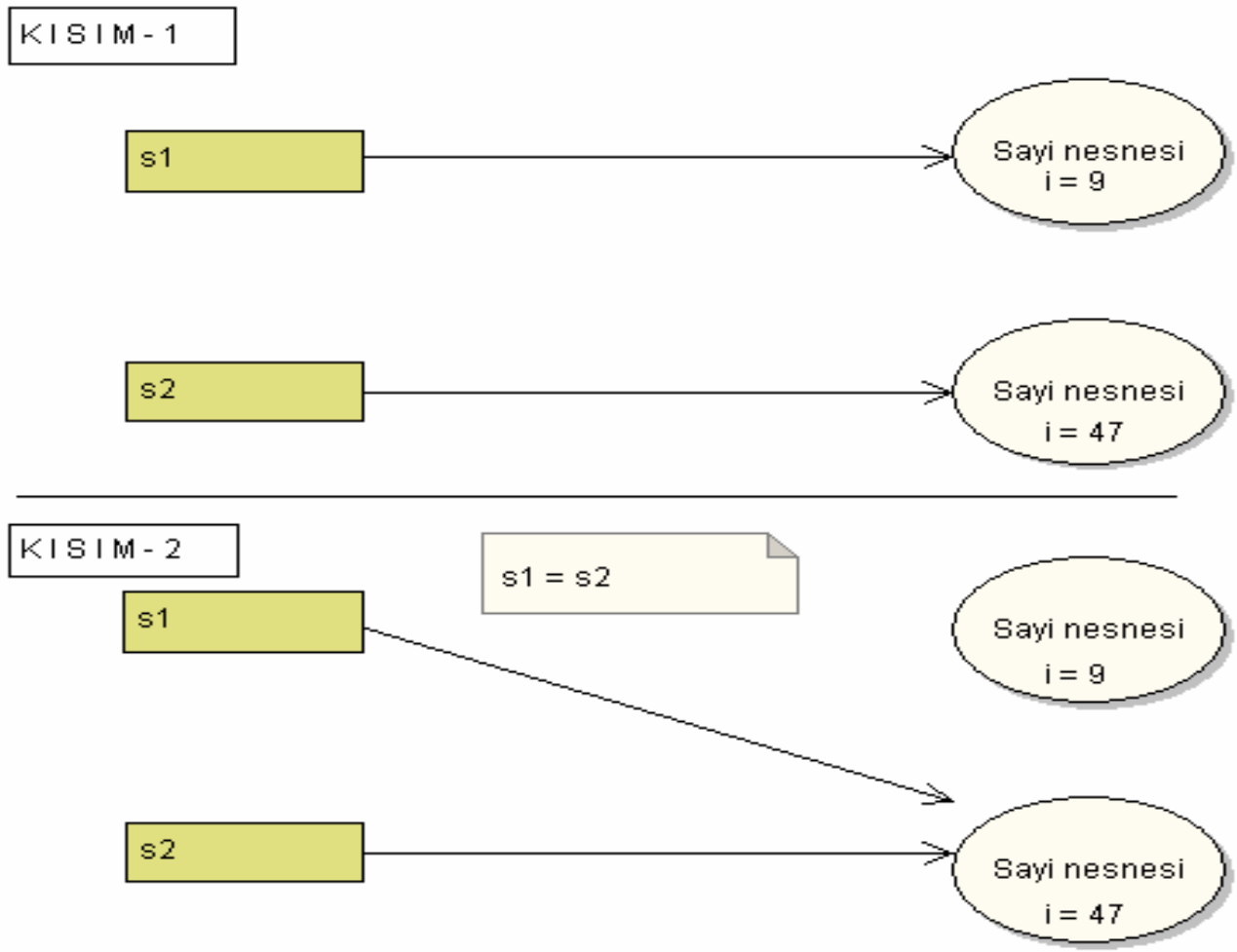
Nesneler ve Atamalar

NesnelerdeAtama.java

Sonuç

- `1: s1.i: 9, s2.i: 47`
- `2: s1.i: 47, s2.i: 47`
- `3: s1.i: 27, s2.i: 27`

Şekil


KISIM - 1
s1
Sayi nesnesi
i = 9
s2
Sayi nesnesi
i = 47
KISIM - 2
s1 = s2
s1
Sayi nesnesi
i = 9
s2
Sayi nesnesi
i = 47

Dosya İsimleri

- Fiziksel dosya ismi ile **`public`** sınıfın ismi aynı olmalı.

Yordam (Method) Çağırımları

- Yordamlar parametre alırlar.
- Alınan bu paremetreler ile yordam içerisinde işlemler gerçekleşir.
- Peki yordamlara parametre olarak ne gitmektedir ?
  - Nesnenin kendisi mi ?
  - Yoksa nesneye bağlı referans mı ?

 ***IlkelPas.java***

 ***Pas.java***

java Operatörleri

- Operatörler programlama dillerinin en temel işlem yapma yeteneğine sahip simgesel isimlerdir.

  - Aritmetik Operatör
  - İlişkisel Operatör
  - Mantıksal Operatörler
  - Bit düzeyinde (*bitwise*) Operatörler

java Operatörleri

- Operatörler bir veya daha fazla değişken üzerinden işlemler gercekleştirirler.
  - İşlem gerçekleştirmek için tek bir değişkene ihtiyaç duyan operatörlere tekli operatör (unary operator)
  - İşlem gerçekleştirmek için iki değişkene ihtiyaç duyan operatörlere ikili operatör (binary operator)
  - İşlem gerçekleştirmek için üç adet değişkene ihtiyaç duyan operatörlere ise üçlü operatör (ternary operator) denir (bir adet var).

Aritmetik Operatörler

Operatör	Kullanılış	Açıklama
+	`değişken1 + değişken2`	değişken1 ile değişken2 yi toplar
-	`değişken1 - değişken2`	değişken1 ile değişken2 yi çıkarır
*	`değişken1 * değişken2`	değişken1 ile değişken2 yi çarpar
/	`değişken1 / değişken2`	değişken1 ,değişken2 tarafından bölünür
%	`değişken1 % değişken2`	değişken1 in değişken2 tarafından bölümünden kalan hesaplanır.

AritmetikOrnek.java

“+” ve “–” Operatörleri

Operatör	Kullanılış Şekli	Açıklama
+	+ değişken	Eğer değişken *char*, *sekizli* (*byte*) veya *short* tipinde ise *int* tipine dönüştürür.
-	- değişken	Değişkenin değerini negatif yapar (-1 ile çarpar).

OperatorTest.java

Dönüştürme *(Casting)* İşlemi

- Bir temel (*primitive*) tip, diğer bir temel tipe dönüştürülebilir, fakat oluşacak değer kayıplarından kodu yazan kişi sorumludur .

IlkelDonusum.java

String (+) Operatörü

- “+” operatörü **String** tiplerde birleştirme görevi görür.
- Eğer bir ifade **String** ile başlarsa , onu takip eden tiplerde otomatik olarak String nesnesine dönüştürülür.

OtomatikCevirim.java

Uygulamanın Çıktısı

- Sonuc = 012
- **String** bir ifadeden sonra gelen tamsayılar görüldüğü üzere toplanmadı.
- Direk **String** nesnesine çevrilip ekrana çıktı olarak gönderildiler.

Bir Arttırma ve Azaltma

- Java dilinde C dilinde olduğu gibi birçok kısaltmalar vardır.
- Bu kısaltmalar hayatı bazen daha güzel bazen ise çekilmez kılabilir.

Bir Arttırma ve Azaltma Tablosu

Operatör	Kullanılış Şekli	Açıklama
++	**değişken++**	Önce değişkenin değerini hesaplar sonra değişkenin değerini bir arttırır.
++	**++değişken**	Önce değişkenin değerini arttırır sonra değişkenin değerini hesaplar.
--	**değişken--**	Önce değişkenin değerini hesaplar sonra değişkenin değerini bir azaltır.
--	**--değişken**	Önce değişkenin değerini azaltır sonra değişkenin değerini hesaplar.

Uygulama

OtomatikArtveAz.java

Uygulamanın Çıktısı

```
i    : 1
++i  : 2
i++  : 2
i    : 3
--i  : 2
i--  : 2
i    : 1
```

İlişkisel Operatörler

- İlişkisel operatörler iki değeri karşılaştırarak bu değerler arasındaki mantıksal ilişkiyi hesaplarlar.
- Örneğin iki değer birbirine eşit değilse "`5==8`"
- Bu ilişki çerçevesinde hesaplanan değer ***false*** olacaktır.

İlişkisel Operatörler Tablosu

Operatör	Kullanılışı	true değeri döner eğer ki...
`>`	`değişken1 > değişken2`	değişken1 , değişken2'den büyükse
`>=`	`değişken1 >= değişken2`	değişken1 , değişken2'den büyükse veya eşitse
`<`	`değişken1 < değişken2`	değişken1 , değişken2'den küçükse
`<=`	`değişken1 <= değişken2`	değişken1 , değişken2'den küçükse veya eşitse
`==`	`değişken1 == değişken2`	değişken1 , değişken2'ye eşitse
`!=`	`değişken1 != değişken2`	değişken1 , değişken2'ye eşit değilse

Uygulama

IliskiselDeneme.java

Nesnelerin Karşılaştırılması

- Nesnelerin eşit olup olmadığı (= =) veya (! =) operatörleri ile test edilebilir mi ?

Denklik.java (*)

Uygulamanın Çıktısı

- `false`
- `true`

Uygulama

- Peki bir önceki örneği **Integer** nesneleri yerine temel tip olan **int** tipini kullansaydık sonuç nasıl olurdu?

IntIcinDenklik.java

Mantıksal Operatörler

- Mantıksal operatörler birden çok karşılaştırma işleminin birleştirip tek bir koşul ifadesi haline getirilmesi için kullanılır.

Mantıksal Operatörler Tablosu

Operatör	Kullanılış Şekli	true değeri döner eğer ki.....
`&&`	`değişken1 && değişken2`	Eğer hem **`değişken1`** hemde **`değişken2`** **true** ise ; (**`değişken2'yi`** duruma göre hesaplar*)
`||`	`değişken1 || değişken2`	**`değişken1'in`** veya **`değişken2'in`** **true** olması ;(**`değişken2'yi`** duruma göre hesaplar*)
`!`	`! değişken`	Eğer **`değişken`** **false** ise
`&`	`değişken1 & değişken2`	Eğer hem **`değişken1`** hemde **`değişken2`** **true ise ;**
`|`	`değişken1 | değişken2`	**`değişken1'in`** veya **`değişken2'in`** **true** olması ;
`^`	`değişken1 ^ değişken2`	**Eğer `değişken1` ve `değişken2`** birbirlerinden farklı ise; ör: **`değişken1`** **true ,`değişken2` false ise***

Uygulama

KosulOp.java

Uygulamanın Çıktısı

```
(a < b) && (c < d) --> false
(a < b) || (c < d) --> true
! (a < b) --> false
(a < b) & (c < d) --> false
(a < b) | (c < d) --> true
(a < b) ^ (c < d) --> true
```

Kısa Yollar

- `i = i + 1 ;` yerine.
- `i += 1 ;` kullanılabilir.

- `i = i * 1 ;` yerine
- `i *= 1 ;` kullanılabilir.

- .....

Kontrol İfadeleri

- Kontrol ifadeleri bir uygulamanın hangi durumlarda ne yapması gerektiğini belirtir.

- Java programlama dilinde toplam 4 adet kontrol ifade çeşidi bulunur.

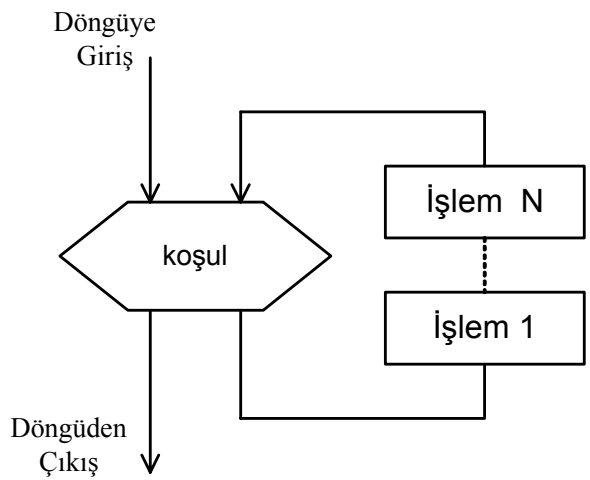
Kontrol İfadeleri Tablosu

İfade Tipi	Anahtar Kelime
Döngü	`while, do-while , for`
Karar verme	`if-else, switch-case`
Dallandırma	`break, continue, label, return`
İstisna yakalama	`try-catch-finally, throw`

Döngü - `while`

- **`while`** ifadesi, çalışması istenen kod bloğunu, durum **true** ifadesini bulana kadar devamlı olarak çalıştırır.

```
while (koşul) {
    ...
  çalışması istenen kod bloğu
}
```



Uygulama

WhileOrnek.java

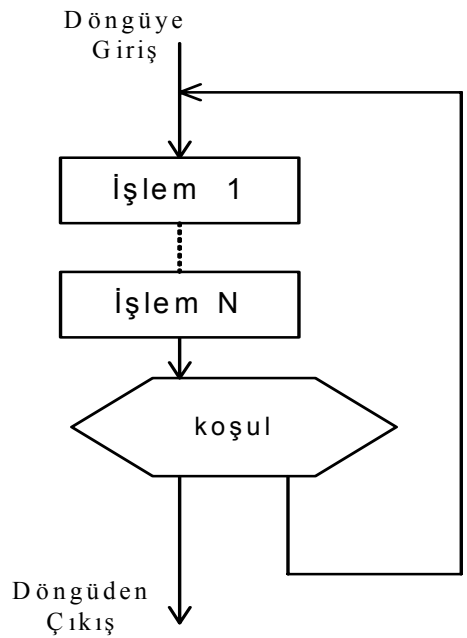
Uygulamanın Çıktısı

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
Sayma islemi tamamlandi.
```

Döngüleme – `do while`

- ***`do-while`*** ifadesi, koşulu en yukarıda değil de en aşağıda hesaplar.
- Böylece ***`do-while`*** ifadesinde durum ***false*** olsa bile çalışması istenen kod bloğuna en az bir kere girilir.

WhileDoOrnek.java



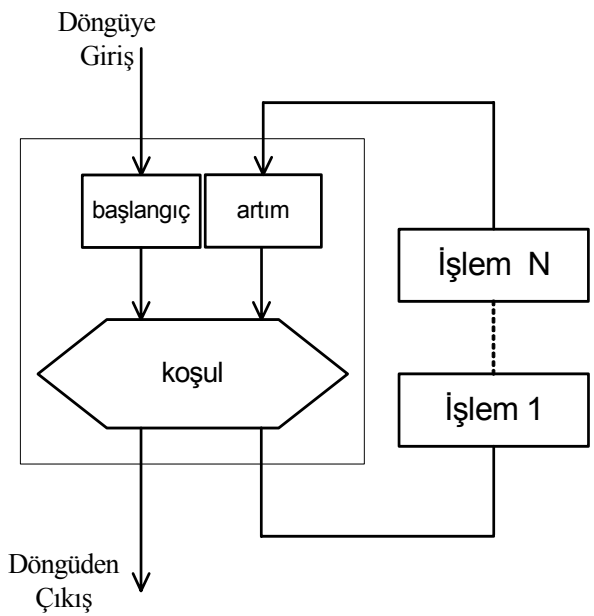
`while` Döngüsü Kullanırken Dikkat Edilmesi Gereken Hususlar

1. Döngü kontrol değişkenine uygun bir şekilde değer atandığına dikkat edilmeli.
2. Döngü durumunun ***true*** ile başlamısına dikkat edilmeli.
3. Döngü kontrol değişkeninin uygun bir şekilde güncellendiğinden emin olunması gerekir (sonsuz döngüye girmemesi için) .

Döngüleme – `for` ifadesi

- Döngünün ne zaman başlayacağı ve ne zaman biteceği en başta belirtilmiştir.

```
for (başlangıç; koşul; artış) {
   çalışması istenen kod bloğu
}
```



Uygulama

ForOrnek.java

`for` İle Sonsuz Döngü

```
for ( ; ; ) {     // sonsuz döngü
     ...
}
```

Uygulamanın Çıktısı

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

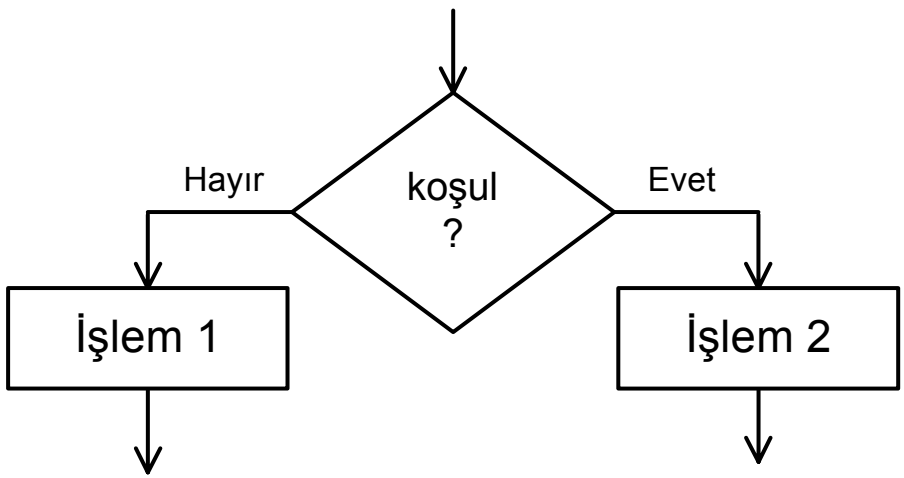
`for` - Çoklu Değişken

```
public class ForOrnekVersiyon2 {

  public static void main(String args[]) {

    for ( int i = 0 , j = 0 ; i < 20 ; i++ , j++ )
    {
       i *= j ;
       System.out.println("i = " + i + " j = " + j);
    }
  }
}
```

```
i = 0  j = 0
i = 1  j = 1
i = 4  j = 2
i = 15 j = 3
i = 64 j = 4
```

Karar Verme - `if`

```
if (koşul) {
  durum true olduğunda çalışması istenen kod bloğu
} else {
  durum false olduğunda çalışması istenen kod bloğu
}
```



Uygulama

IfElseTest.java

Üçlü `if-else`

`boolean-ifade ? deger0 : deger1`

- Eğer *boolean* ifade *true* ise **`değer0`** hesaplanır , eğer *boolean* ifade false ise **`deger1`** hesaplanır.

Kısa Devre

- **if** ifadesinde eğer **VE(&&)** işlemi kullanılmış ise ve ilk değerden **false** dönmüş ise ikinci değer kesinlikle hesaplanmaz çünkü bu iki değerin sonucunun **VE(And)** işlemine göre **true** dönmesi imkansızdır.

- Kısa devre özelliği sayesinde uygulamalar gereksiz hesaplamalardan kurtulmuş olur.

KisaDevre.java

Karar Verme - **switch**

```
switch(tamsayı) {
  case uygun-tamsayı-deger1 : çalışması istenen kod bloğu; break;
  case uygun-tamsayı-deger2 : çalışması istenen kod bloğu; break;
  case uygun-tamsayı-deger3 : çalışması istenen kod bloğu; break;
  case uygun-tamsayı-deger4 : çalışması istenen kod bloğu; break;
  case uygun-tamsayı-deger5 : çalışması istenen kod bloğu; break;
          // ...
  default: çalışması istenen kod bloğu ;
}
```

Uygulama 1

```
public class AylarSwitchTest {

    public static void main(String[] args) {

        int ay = 8;
        switch (ay) {
            case 1:  System.out.println("Ocak"); break;
            case 2:  System.out.println("Subat"); break;
            case 3:  System.out.println("Mart"); break;
            case 4:  System.out.println("Nisan"); break;
            case 5:  System.out.println("Mayis"); break;
            case 6:  System.out.println("Haziran"); break;
            case 7:  System.out.println("Temmuz"); break;
            case 8:  System.out.println("Agustos"); break;
            case 9:  System.out.println("Eylul"); break;
            case 10: System.out.println("Ekim"); break;
            case 11: System.out.println("Kasim"); break;
            case 12: System.out.println("Aralik"); break;
        }
    }
}
```

Uygulama 2

```
public class AylarSwitchTestNoBreak {

    public static void main(String[] args) {

        int ay = 8;
        switch (ay) {
           case 1:  System.out.println("Ocak");
           case 2:  System.out.println("Subat");
           case 3:  System.out.println("Mart");
           case 4:  System.out.println("Nisan");
           case 5:  System.out.println("Mayis");
           case 6:  System.out.println("Haziran");
           case 7:  System.out.println("Temmuz");
           case 8:  System.out.println("Agustos");
           case 9:  System.out.println("Eylul");
           case 10: System.out.println("Ekim");
           case 11: System.out.println("Kasim");
           case 12: System.out.println("Aralik");
        }
    }
}
```

Uygulama 2 - Ekran Çıktısı

```
Agustos
Eylul
Ekim
Kasim
Aralik
```

Uygulama 3

```
public class AylarSwitchDefaultTest {

  public static void main(String[] args) {

   int ay = 25;
   switch (ay) {
     case 1:  System.out.println("Ocak"); break;
     case 2:  System.out.println("Subat"); break;
     case 3:  System.out.println("Mart"); break;
     case 4:  System.out.println("Nisan"); break;
     case 5:  System.out.println("Mayis"); break;
     case 6:  System.out.println("Haziran"); break;
     case 7:  System.out.println("Temmuz"); break;
     case 8:  System.out.println("Agustos"); break;
     case 9:  System.out.println("Eylul"); break;
     case 10: System.out.println("Ekim"); break;
     case 11: System.out.println("Kasim"); break;
     case 12: System.out.println("Aralik"); break;
     default: System.out.println("Heyoo,Aranilan Kosul" +
                                 "Bulunamadi!!");
    }
  }
}
```

Dallandırma İfadeleri

- Java programlama dilinde dallandırma ifadeleri toplam 3 adettir.
  - **`break`** ifadesi
  - **`continue`** ifadesi
  - **`return`** ifadesi

`break` İfadesi - Etiketsiz

BreakTest.java

Uygulama Çıktısı

```
i =0
i =1
i =2
i =3
i =4
i =5
i =6
i =7
i =8

  Donguden cikti
```

`break` İfadesi - Etiketli

BreakTestEtiketli.java

Uygulama Çıktısı

```
i =0
i =1
i =2
i =3
i =4
i =5
i =6
i =7
i =8
```

`continue` İfadesi - Etiketsiz

ContinueTest.java

Uygulama Çıktısı

```
i =0
i =1
i =2
i =3
i =4
i =5
i =6
i =7
i =8 → 9 yok
i =10
i =11
i =12
i =13
i =14
i =15
i =16
i =17
i =18
i =19
i =20
i =21
i =22
i =23
i =24
i =25
i =26
i =27
i =28
i =29
      Donguden cikti
```

`continue` İfadesi - Etiketli

ContinueTestEtiketli.java

Uygulama Çıktısı

```
i =0
i =1
i =2
i =0
i =1
i =2
i =0
i =1
i =2
i =0
i =1
i =2
i =0
i =1
i =2
i =0
i =1
i =2
i =0
i =1
i =2
```

`return` İfadesi - Etiketli

- Sadece **`return`** anahtar kelimesi kullanarak yordamların içerisini tavizsiz bir şekilde terk edelebilir.

Sorular ...