Temel Operatörler

Bir işleme yola açan ve işlem sonucunda bir değer üretilmesini sağlayan atomlara **operatör** denir. Operatörler 3(üç) biçimde sınıflandırılabilabilirler:

1. İşlevlerine söre sınıflandırma
2. Operand sayılarına göre sınıflandırma
3. Operatörün konumuna göre sınıflandırma

1. İşlevlerine göre sınıflandırma: Bu sınıflandırmada operatörün ne amaçla kullanıldığına bakılır. Buna göre işlevlerine göre operatörleri aşağıdaki gibi gruplandırabiliriz:

- Aritmetik operatörler (arithmetic operators)
- Karşılaştırma operatörleri (comparison operators)
- Mantıksal operatörler (logical operators)
- Bitsel operatörler (bitwise operators)
- Özel amaçlı operatörler (special purpose operators)

2. Operand sayılarına göre sınıflandırma: Operatörle işleme sokulan ifadelere **operand** denir. Operand sayılarına göre operatörleri aşağıdaki gibi gruplandırabiliriz:

- Tek operandlı operatörler (unary operators)
- İki operandli operatörler (binary operators)
- Üç operandlı operatörler (ternary operators)

3. Operatörün konumuna göre sınıflandırma: Bu sınıflandırmada operatörün operand(lar)ının neresinde olduğuna bakılır. Üç şekilde gruplandırılabilir:

- Önek operatörler (prefix operators)
- Araek operatörler (infix operators)
- Sonek operatörler (postfix operators)

Operatörlerin teknik olarak tanımlanabilmesi için bu üç sınıflandırmada nereye düştüğü belirtilmelidir. Bundan sonra operatöre ilişkin ilave bazı özellikler belirtilmelidir. Örneğin "+ operatörü iki operandli araek durumunda bir aritmetik operatördür". Ya da örneğin "! operatörü tek operandı önek durumunda mantıksal bir operatördür. Ayrıca bu operatörün operandı boolean türden bir ifade olmalıdır ve ürettiği değer de boolean türdendir" gibi…

Farklı işlevleri olan operatörlere ilişkin atomlar aynı olabilmektedir. Örneğin + karakteri hem toplama operatörü hem de işaret operatörü olarak kullanılmaktadır.

Operatörler Arasındaki Öncelik İlişkisi

Her operatörün diğerine göre bir öncelik durumu (precedence) vardır. Örneğin * operatörü + operatöründen daha yüksek önceliklidir. Operatörlerin bazıları eşit öncelikte olabilirler. İşlemler genel olarak öncelik sırasına göre yapılır. Örneğin:

"i" yapılan işlemi temsil etmek üzere;

```
a = b + c * d;
```

işleminin yapılışı

i1: c * d
i2: b + i1
i3: a = i2
şeklindedir.

"( )" parantezler kullanılarak önceliklendirme sağlanabilir. Örneğin:

```
a = (b + c) * d;
```

işlemi

i1: b + c
i2: i1 * d
i3: a = i2
şeklinde yapılır.

Operatörler arasındaki öncelik ilişkisi **operatör öncelik tablosu** denilen bir tabloyla betimlenir. Örneğin:

Operatör	**İlişkisi**
( )	Soldan sağa
* /	Soldan sağa
+ -	Soldan sağa
=	Sağdan sola

Öncelik tablosunda yukarıdaki satırlar aşağıdaki satırlara göre yüksek önceliklidir. Aynı satırdakiler eşit önceliklidir. Eşit öncelikli operatörler ifade içerisindeki konumlarına göre **soldan sağa (left associative)** ya da **sağdan sola (right associative)** öncelikli olarak yapılırlar. Örneğin:

```
a = b * c / d;
```

Burada * ve / operatörleri eşit öncelikli olduğundan ve bu operatörler soldan sağa öncelikli olduğundan işlemler

i1: b * c
i2: i1 / d
i3: a = i2
şeklinde yapılır.

***, /, + ve – operatörleri**

Bu operatörler iki operandlı araek durumunda aritmetik operatörlerdir. Matematikteki klasik 4(dört) işlemi yaparlar. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b, c;

		a = 10;
		b = 5;

		c = a + b;

		System.out.println(c);

		c = a * b;
		System.out.println(c);

		c = a / b;
		System.out.println(c);

	}
}
```

% operatörü

Bu operatöre "mod operatörü" de denmektedir. Operatör sol taraftaki operandın sağ taraftaki operanda bölümünden elde edilen kalan değerini üretir.

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b, c;

		a = 10;
		b = 5;

		c = a % b;

		System.out.println(c);
	}
}
```

% operatörünün ürettiği değer için bölme işlemi yapıldığından operatör öncelik tablosunda * ve / operatörü ile soldan sağa öncelikli grupta bulunmaktadır.

Operatör	İlişkisi
( )	Soldan sağa
* / %	Soldan sağa
+ -	Soldan sağa
=	Sağdan sola

İşaret + ve İşaret – operatörleri

İşaret + ve işaret – operatörleri tek operandlı önek durumunda aritmetik operatörlerdir. İşaret – operatörü operandının negatif değerini üretir.

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b;

		a = 10;

		b = -a;

		System.out.println(a);
		System.out.println(b);
	}
}
```

İşaret + operatörü operandı ile aynı değeri üretir. Yani aslında hiçbir işlem yapmaz.

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b;

		a = 10;

		b = +a;

		System.out.println(a);
		System.out.println(b);
	}
}
```

İşaret + ve işaret – operatörleri öncelik tablosunun ikinci düzeyinde sağdan sola grupta bulunurlar.

Operatör	İlişkisi
( )	Soldan sağa
+ -	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
=	Sağdan sola

++ ve -- Operatörleri

++ operatörüne artırma (increment) -- operatörüne eksiltme (decrement) operatörü denir. Bu operatörler tek operandlı önek ve sonek olarak kullanılabilen aritmetik operatörlerdir. ++ operatörü operandın değerini 1(bir) artırır, -- operatörü operandın değerini 1(bir) eksiltir. Bu operatörler öncelik tablosunun ikinci düzeyinde sağdan sola grupta bulunurlar.

Operatör	İlişkisi
( )	Soldan sağa
+ - ++ --	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
=	Sağdan sola

Bu operatörlerin önek ve sonek kullanımları farklıdır. Bu operatörlerin hem önek hem de sonek kullanımları tabloda belirtilen öncelikle yapılır. Ancak önek kullanımda sonraki işleme değişkenin artırılmış ya da eksiltilmiş değeri sokulurken, sonek kullanımda değişkenin artırılmamış ya da eksiltilmemiş değeri sokulur. Yani ++ ve -- operatörlerinin önek kullanımlarında ürettikleri değerler sırasıyla artırılmış ve eksiltilmiş değerlerdir, sonek kullanımlarında ürettikleri değerler de sırasıyla artırılmamış ve eksiltilmemiş değerlerdir. Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b;

		a = 10;

		b = ++a;

		System.out.println(a); // 11
		System.out.println(b); // 11
	}
}
```

Örneğin:

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b;

		a = 10;

		b = a++;

		System.out.println(a); // 11
		System.out.println(b); // 10
	}
}
```

++ ve -- operatörlerinin operandları değişken olmalıdır. Yani örneğin:

```
b = 8++; //error
```

ifadesinde ++ operatörünün operandı sabit olduğundan ifade geçersizdir.

Anahtar Notlar: C/C++ dillerinde ++ ve -- operatörleri ile bu operatörlerin operandlarının aynı ifadede bulunması tanımsız davranışa (undefined behaviour) yol açmaktadır. Java' da herhangi bir sorun bulunmamaktadır. Derleyicinin tanımsız davranışı C/C++ standartlarına göre kısaca ne olacağı belli olmayan durumlar için kullanılmaktadır. Yani bu dillerde "programcı tanımsız davranışa yol açacak kodlardan kaçınmalıdır" anlamına gelmektedir. Java' da tanımsız davranışa yol açacak hiç bir durum yoktur. Yani Java' da bir ifade ya geçerli ya da geçersiz (error) olmaktadır.

"Bilindiği gibi Java' da atomlar arasında istenildiği kadar boşluk bırakılabilir. Atomlar istenildiği kadar bitişik yazılabilir. Anahtar sözcükler ve değişkenler bitişik yazılamaz" kuralları geçerlidir. Peki aşağıdaki gibi bir ifadeyi derleyici nasıl algılayacaktır?
a = b+++c;
Yani derleyici bu ifade için, "a = b + ++c;" biçiminde mi yoksa "a = b++ + c;" biçiminde mi işlem yapacaktır? Burada derleyici derleme sırasında anlamlı en uzun ifadeyi seçer. Yani "a = b++ + c" biçiminde işlem yapılacaktır. Örnek bir test kodu aşağıdaki gibi yazılabilir:

```
package csd;

class App {
	public static void main(String [] args)
	{
		int a, b, c;

		a = 10;
		b = 8;

		c = a+++b; // c = a++ + b;

		System.out.println(a); //11
		System.out.println(b); //8
		System.out.println(c); //18
	}
}
```

Karşılaştırma Operatörleri

Toplam 6(altı) adet karşılaştırma operatörü vardır. Bu operatörler iki operandlı araek durumunda operatörlerdir. Karşılaştırma operatörleri boolean türden değer üretirler. Bu operatörler öncelik tablosunda artimetik operatörlerden daha düşük öncelikli gruptadır.

Operatör	İlişkisi
( )	Soldan sağa
+ - ++ --	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
><>= <=	Soldan sağa
== !=	Soldan sağa
=	Sağdan sola

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int a, b;

		a = 10;
		b = 8;

		boolean result;

		result = a >= b;

		System.out.println(result); //true
	}
}
```

Bu operatörlerden == operatörü eşitlik karşılaştırması yapar. Operandlar birbirine eşitse true değilse false değerini üretir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int a, b;

		a = 8;
		b = 8;

		boolean result;

		result = a == b;

		System.out.println(result); //true
	}
}
```

Benzer şekilde != operatörü operandlarının eşitsizliği durumunda true eşitliği durumunda false değerini üretir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int a, b;

		a = 8;
		b = 8;

		boolean result;

		result = a != b;
```

```
            System.out.println(result); //false

        }
}
```

Yani bu iki operatör birbirlerinin mantıksal olarak tersleridir.

Benzer şekilde < operatörünün mantıksal tersi >= operatörü ve > operatörünün mantıksal tersi de <= operatörüdür.

Mantıksal Operatörler

Java' da 3(üç) adet mantıksal operatör vardır. Bunlar

&& → mantıksal ve (logical AND)
|| → mantıksal veya (logical OR)
! → mantıksal değil (logical NOT)

operatörleridir.

&& ve || operatörleri iki operandlı araek durumunda, ! operatörü tek operandlı önek durumunda operatörlerdir. Mantiksal operatörlerin operandları boolean türden olmak zorundadır. Ürettikleri değerler de boolean türdendir. Ürettikleri değerler şöyledir:

&& ve || operatörleri

A	**B**	**A && B**	**A || B**
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

! operatörü

A	**!A**
True	False
False	True

&& ve || operatörleri öncelik tablosunda karşılaştırma operatörlerinden daha düşük öncelikli soldan sağa grupta bulunmaktadırlar. ! operatörü öncelik tablosunun ikinci düzeyinde sağdan sola grupta bulunmaktadır.

Operatör	**İlişkisi**
( )	Soldan sağa

+ - ++ -- !	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
><>= <=	Soldan sağa
== !=	Soldan sağa
&&	Soldan sağa
||	Soldan sağa
=	Sağdan sola

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		boolean result;

		result = Sample.foo() && Sample.bar();

		System.out.println(result);
	}
}

class Sample
{
	public static boolean foo()
	{
		System.out.println("foo");

		return true;
	}

	public static boolean bar()
	{
		System.out.println("bar");

		return false;
	}
}
```

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		boolean result;

		result = Sample.bar() || Sample.foo();
		System.out.println(result);
	}
}
class Sample
{
	public static boolean foo()
	{
		System.out.println("foo");
```

```
		return true;
	}

	public static boolean bar()
	{
		System.out.println("bar");
		return false;
	}
}
```

&& ve || operatörlerinin kısa devre (short circuit) özellikleri vardır. Bu operatörler klasik öncelik kurallarına uymazlar. Bu operatörlerin sağ tarafında ne olursa olsun önce sol tarafındaki ifade yapılır ve bitirilir. Duruma göre sağ tarafındaki ifade yapılır. "Mantıksal AND" ve "mantıksal OR" işlemleri kısaca şöyle tanımlanabilir:

Mantıksal AND: İfadelerden her ikisi birden true ise sonuç true, diğer bütün durumlarda false.

Mantıksal OR: İfadelerden her ikisi birden false ise işlem false, diğer bütün durumlarda true.

Bu tanımlara göre, && operatörü için sol tarafındaki ifade false ise sağ tarafı ne olsun sonucu etkilemeyecektir. Benzer şekilde || operatörü için de sol tarafındaki ifade true ise sağ tarafı ne olursa olsun sonuç değişmez. Bu sebeple && operatörünün sol tarafı false ise sağ tarafı hiç yapılmaz, || operatörünün sol tarafı true ise sağ tarafı hiç yapılmaz.

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		boolean result;

		result = Sample.bar() && Sample.foo();

		System.out.println(result);
	}
}

class Sample
{
	public static boolean foo()
	{
		System.out.println("foo");

		return true;
	}

	public static boolean bar()
	{
		System.out.println("bar");

		return false;
	}
}
```

Görüldüğü gibi bar metodunun geri dönüş değeri false olduğundan kısa devre özelliğinden dolayı foo metodu çağrılmamaktadır.

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
```

```
	{
		boolean result;

		result = Sample.foo() || Sample.bar();

		System.out.println(result);
	}
}

class Sample
{
	public static boolean foo()
	{
		System.out.println("foo");

		return true;
	}

	public static boolean bar()
	{
		System.out.println("bar");

		return false;
	}
}
```

Aslında kısa devre özelliği farklı bir sonucun çıkmasına yol açmaz. Öncelik tablosu tam olarak uygulanırsa çıkacak sonuç aynıdır. Kısa devre özelliği ile bu sonuç daha hızlı elde edilir.

Java' da & ve | operatörleri bit düzeyinde AND ve OR işlemlerini yapar. Bu operatörler iki operandlı araek durumunda soldan sağa öncelik grubunda operatörlerdir.

Operatör	İlişkisi
( )	Soldan sağa
+ - ++ -- !	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
><>= <=	Soldan sağa
== !=	Soldan sağa
&	Soldan sağa
|	Soldan sağa
&&	Soldan sağa
||	Soldan sağa
=	Sağdan sola

Bu operatörler sayının karşılıklı bitlerini AND ve OR işlemlerine sokarlar. Bu operatörlerin Java' da asıl kullanımları bit işlemleri olmasına karşın boolean türden operandlarla da işleme sokulabilir. & ve | operatörlerinin kısa devre özelliği yoktur. Bu sebeple boolean türden operandlar ile kısa devre özelliği olmayan mantıksal AND ve OR operatörleri gibi de kullanılabilirler.

.
! operatörü tek operandlı önek durumunda bir operatördür. ! operatörünün operandı boolean türden olmak zorundadır. Bu operatör operandının mantıksal olarak tersini üretir. Yani operandının değeri true ise false, false ise true değerini üretir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		boolean a, b;

		a = true;

		b = !a;

		System.out.println(a); // true
		System.out.println(b); // false
	}
}
```

Atama Operatörü

Atama operatörü iki operandlı araek durumunda sağdan sola öncelik gurubunda bulunan operatördür. Bu operatör en düşük öncelikli operatördür.

Operatör	İlişkisi
( )	Soldan sağa
+ - ++ -- !	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
><>= <=	Soldan sağa
== !=	Soldan sağa
&	Soldan sağa
|	Soldan sağa
&&	Soldan sağa
||	Soldan sağa
=	Sağdan sola

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int a, b;
```

```
		b = 10;

		a = b;

		System.out.println(a); //10
		System.out.println(b); //10
	}
}
```

Atama operatörünün ürettiği değer sol taraftaki değişkene atanan değerdir.

İşlemli Atama Operatörleri

Java' da bir grup *=, /=, +=, -= biçiminde işlemli atama operatörleri vardır. Bu operatörler iki operandlı araek durumunda sağdan sola öncelik grubunda operatörlerdir. Bu operatörler atama operatörü ile aynı öncelik grubunda bulunurlar.

Operatör	İlişkisi
( )	Soldan sağa
+ - ++ -- !	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
><>= <=	Soldan sağa
== !=	Soldan sağa
&	Soldan sağa
|	Soldan sağa
&&	Soldan sağa
||	Soldan sağa
= *= /= += -=	Sağdan sola

a <op>= b ifadesi a = a <op> b ile tamamen aynı anlamdadır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int a, b;

		a = 10;
		b = 8;

		a += b; // a = a + b
```

```
            System.out.println(a); //18
            System.out.println(b); //8
        }
}
```

Java programlama dilinin tüm operatörlerine ilişkin öncelik tablosu aşağıdaki gibidir.

Operatör	İlişkisi
( ) . [] new	Soldan sağa
+ - ++ -- ! ~ ( )	Sağdan sola
* / %	Soldan sağa
+ -	Soldan sağa
<< >> >>>	Soldan sağa
> < >= <= instanceof	Soldan sağa
== !=	Soldan sağa
&	Soldan sağa
^	Soldan sağa
|	Soldan sağa
&&	Soldan sağa
||	Soldan sağa
?:	Sağdan sola
= *= /= %= += -= <<= >>= &= ^= |= >>>=	Sağdan sola

Bu bölümde anlatılmayan operatörler için birtakım konuların da bilinmesi gerektiğinden bu operatörler daha sonra ele alınacaktır.

Etkisiz İfadeler

Java' da bir durum değişikliğine yol açmayan kodlara etkisiz kodlar (code has no effect) denir. Etkisiz kod oluşturmak Java'da genel olarak geçersizdir. Etkisiz kodlar derleme zamanında hata (error) oluşturur.

`a * b;  //error`: Etkisiz kod. Bu kod olmasa da programda değişen bir şey olmaz.

Ancak

```
++a;   //Geçerli
foo() // Geçerli
```

Şüphesiz metod çağrılması bir etkiye yol açmaktadır.
Anahtar Notlar: C/C++ dillerinde etkisiz kodlar genel olarak hata (error) oluşturmaz.

Noktalı Virgülün İşlevi

Noktalı virgül ifadeleri birbirinden ayırmak için kullanılmaktadır. Noktalı virgül ile sonlandırılan ifadeler birbirinden bağımsızdır. Eğer noktalı virgül unutulursa önceki ifade ile sonraki ifade tek bir ifade olarak ele alınır. Bu da çoğu zaman hata (error) oluşturur. Örneğin:

```
a = 10 //error
b = 20;
```

Burada ifadenin tamamı anlamsız olduğundan hata oluşturur.

İfadeleri sonlandıran bu tür atomlara sonlandırıcı (terminator) denmektedir. Bazı dillerde ENTER karakteri sonlandıcı atom olarak kullanılır. Bu dillerde her satıra tek bir ifade yazılmak zorundadır. Örneğin BASIC programlama dili bu şekildedir. Java 'da sonlandırıcı noktalı virgüldür.