Farklı Türlerin Birbirine Atanması (Dönüştürülmesi)

Bir atama işleminde kaynak (source) ve hedef (destination) türler vardır. En normal durum bu türlerin aynı olmasıdır. Atama işleminde kaynak tür derleyici tarafından önce otomatik olarak (implicit) hedef türe dönüştürülür. Daha sonra atama yapılır. Yani T1 türünden T2 türüne doğrudan atama işlemi yapılabilmesi için T1 türünden T2 türüne otomatik dönüştürmenin (implicit conversion) var olması gerekir. Java' da genel olarak küçük türden büyük türe otomatik dönüştürme vardır. Ancak büyük türden küçük türe otomatik dönüştürme yoktur. Yani küçük türden büyük türe doğrudan atama yapılabilir. Örneğin int türü long türüne doğrudan atanabilir fakat long türü int türüne doğrudan atanamaz. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		long a = 10;
		int b;

		b = a; //error

	}
}
```

Java'da genel olarak bilgi kaybına yol açmayacak dönüşümler otomatik olarak yapılabilmektedir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		long a;
		int b = 10;

		a = b; // geçerli
	}
}
```

long türünün tutabileceği değerler int türünün tutabileceği değerleri de kapsadığından bilgi kaybı olmaz.

Atama işleminde değişkenin içerisindeki değere bakılmaz, yalnızca türlere bakılır. Yani büyük türün içerisinde küçük türün sınırları içerisinde kalabilecek değer olsa bile doğrudan atama yapılamaz. Örneğin:

```
package csd;

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

		a = 8;

		b = a; //error
	}
}
```

Anahtar Notlar: "The Java Language Specification" dökumanında genel olarak küçük türden büyük türe dönüştürme işlemine genişletme dönüştürmesi (widening conversion), büyük türden küçük türe dönüştürme işlemine daraltma dönüştürmesi (narrowing conversion) denilmektedir.

Küçük türden büyük türe atamanın ayrıntıları:

1. Küçük tamsayı türünden büyük tamsayı türüne doğrudan atama yapılabilir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		short a;

		a = 20;

		int b = a; //geçerli
	}
}
```

2. Küçük tamsayı türünden büyük tamsayı türüne atama yapıldığında küçük tamsayı içerisindeki değer pozitifse sayının genişletilen (yüksek anlamlı) bitleri 0(sıfır) ile beslenir. Örneğin:

```
package csd;

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

		a = 0x11FC;

		System.out.printf("a=%d\n", a);

		b = a;

		System.out.printf("b=0x%X\n", b); // 0x11FC
		System.out.printf("b=%d\n", b);
	}
}
```

3. Küçük tamsayı türünden büyük tamsayı türüne atama yapıldığında küçük tamsayı içerisindeki değer negatifse sayınının değerinin korunması için sayının genişletilen (yüksek anlamlı) bitleri 1(bir) ile beslenir. Örneğin:

```
package csd;

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

		a = -2; //0xFFFE

		System.out.printf("a=0x%X\n", a);  //a=0xFFFE

		b = a;

		System.out.printf("b=0x%X\n", b);  //b=0xFFFFFFFE
		System.out.printf("b=%d\n", b);    //b=-2
	}
}
```

4. Hiçbir türden char türüne doğrudan atama yapılamaz. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		char ch;
		byte b;

		b = 10;
		ch = b; //error
	}
}
```

Fakat bu kurala istisna olarak char türünden bir değişkene değer sınırlar içerisinde kalıyorsa int türden bir sabit ifadesi doğrudan atanabilir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		char ch;

		ch = 97;

		System.out.printf("ch=%c\n", ch);
	}
}
```

5. char türünden kendisinden büyük türlere doğrudan atama yapılabilir. Örneğin:

```
package csd;

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

		ch = 'a';
		b = ch;

		System.out.printf("b=%d\n", b);
	}
}
```

6. char türünden short türüne doğrudan atama yapılamaz. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		char ch;
		short b;

		ch = 'a';

		b = ch; //error
```

```
		System.out.printf("b=%d\n", b);
	}
}
```

7. Bütün tamsayı türlerinden gerçek sayı türlerine doğrudan atama (yani dönüştürme) vardır. Örneğin:

```
package csd;

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

		a = 10;

		double b;

		b = a;

		System.out.printf("b=%f\n", b);
	}
}
```

Fakat gerçek sayı türlerinden tamsayı türlerine doğrudan atama (yani dönüştürme) yapılamaz. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		float a;

		a = 10.3F;

		long b = a; //error:
	}
}
```

8. Java'da byte ve short türden sabit yoktur. Ancak int türden bir sabit ifadesi belirttiği sayı hedef türün sınırları içerisinde kalıyorsa byte ve short türlerine doğrudan atanabilir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		byte a;

		a = 20;

		System.out.printf("a=%d\n", a);
	}
}
```

Atanan sabit ifadesinin değeri hedef tütün sınırları dışında kalıyorsa error oluşur. Örneğin:

```
package csd;

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

```
	{
		byte a;

		a = 220;//error

		System.out.printf("a=%d\n", a);
	}
}
```

9. boolean türünden herhangi bir türe, herhangi bir türden boolean türüne doğrudan atama yapılamaz. Yani otomatik dönüştürme yoktur. Örneğin:

```
package csd;

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

		b = true;

		a = b; // error

		System.out.printf("a=%d\n", a);
	}
}
```

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		boolean b;
		int a = 1;

		b = a; //error
	}
}
```

Buna göre hangi türden hangi türe doğrudan atama (implicit conversion) yapılabileceği aşağıdaki tabloyla kesin olarak belirtilebilir:

byte → short, int, long, float, double
short → int, long, float, double
int → long, float, double
long → float, double
char → int, long, float, double
float → double

İşlem Öncesi Otomatik Tür Dönüşümleri

Programlama dillerinde yalnızca değişkenlerin ve sabitlerin değil aynı zamanda her ifadenin de bir türü vardır. Örneğin a'nın bir türü vardır, 8 sabitinin de bir türü vardır, a + 8 ifadesinin de bir türü vardır.

Bir operatörün operandları farklı türlerden olabilir. Derleyici bir operatörle karşılaştığında operandların türlerine bakar. Bunlar farklı türlerdense önce onları aynı türe dönüştürür, ondan sonra işlemi yapar. Özet kural şudur:
"Küçük tür büyük türe dönüştürülür ve sonuç büyük tür türünden çıkar"

Örneğin:

```
int a = 10;
double b = 8.11;
int c;

c = a + b // error
```

Burada a + b işleminden önce a double türüne dönüştürülür, sonra işlem yapılır. Sonuç double türünden çıkar. double türünden int türüne otomatik dönüştürme olmadığından (doğrudan atama yapılamadığından) derleme zamanında hata oluşur.

"Küçük tür büyük türe dönüştürülür ve sonuç büyük tür türünden çıkar" kuralının da bazı ayrıntıları vardır:

1. İşlem öncesinde her iki operand da byte short ya da char türündense (yani int türünden küçükse) önce her iki operand da bağımsız olarak int türüne dönüştürülür, sonuç int türünden çıkar. Bu kurala int türüne yükseltme (integral promotion) denilmektedir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		short b1, b2;
		short result;

		b1 = 10;
		b2 = 20;

		result = b1 + b2; //error: b1 + b2 ifadesi int türden!

		System.out.printf("result=%d\n", result);
	}
}
```

2. Gerçek sayı türleriyle tamsayı türleri işleme sokulursa dönüştürme her zaman gerçek sayı türlerine doğru yapılır. Sonuç dönüştürülen gerçek sayı türünden çıkar. Örneğin:

```
package csd;

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

		a = 10;
		b = 20.456;

		result = a + b;

		System.out.printf("result=%f\n", result);
```

```
	}
}
```

3. Bölme işleminde her iki operand da tam sayı türlerine ilişkinse sonuç tamsayı türünden çıkar. Bölme işlemi yapılır, sonucun noktadan sonraki kısmı atılır. Örneğin:

```
package csd;

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

		a = 10;
		b = 8;

		result = a / b; // dikkat sonucun noktadan sonraki kısmı atılır.

		System.out.printf("result=%f\n", result);
	}
}
```

4. boolean türü hiçbir biçimde diğer türlerle işleme sokulamaz.

```
package csd;

class App {

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

		a = true;
		b = false;

		boolean result = a + b; //error
	}
}
```

5. char türü diğer türlerle işleme sokulabilir.

```
package csd;

class App {

	public static void main(String[] args)
	{
		char ch = 'a';
		int b = 10;
		int val = ch + b;

		System.out.printf("val=%d\n", val);
	}
}
```

Anahtar Notlar: İşlem öncesinde küçük türün büyük türe dönüştürülmesi geçici değişken yoluyla yapılmaktadır. Önce büyük tür türünden geçici bir değişken yaratılır ve küçük türün içerisindeki değer ona atanır. İşleme geçici değişken sokulur, işlem sonrasında geçici değişken yok edilir.

Tür Dönüştürme Operatörü

Tür dönüştürme operatörü tek operandlı önek durumunda bir operatördür. Genel biçimi şöyledir:

(tür) operand

Buradaki parantez operatörü belirtmektedir. Öncelik parantezi değildir. Tür dönüştürme operatörü öncelik tablosunun ikinci düzeyinde sağdan sola grupta bulunur:

Operatör	**İlişkisi**
( )	Soldan Sağa
+ - ++ -- ! (tür)	Sağdan Sola
* / %	Soldan Sağa
+ -	Soldan Sağa
<><= >=	Soldan Sağa
== !=	Soldan Sağa
&&	Soldan Sağa
||	Soldan Sağa
=	Sağdan Sola

Örneğin:

```
a = (double) b / c;

i1: (double) b
i2 = i1 / c
a = i2
```

Dönüştürme işlemi geçici değişken yoluyla yapılmaktadır. Yani önce dönüştürülecek tür türünden geçici bir değişken yaratılır, dönüştürülecek değer ona atanır, sonra işleme geçici değişken sokulur. İşlem sonrasında geçici değişken yok edilir.

Tür dönüştürme operatörü ile yapılan dönüştürmelere “explicit conversion” denilmektedir. Bilindiği gibi otomatik tür dönüştürmelere “implicit conversion” denilmektedir. Tür dönüştürme operatörü ile bazı istisnalar dışında her temel tür her temel türe dönüştürülebilir. Örneğin:

```
package csd;

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

		a = 1000000;
		b = (short)a;

		System.out.printf("b=%d\n", b);
	}
}
```

Tür dönüştürme operatörü ile bazı istisnalar dışında her tür her türe dönüştürülebildiğine göre bilgi kaybı söz konusu olabilmektedir. Bilgi kaybının nasıl oluşacağına ilişkin kurallar belirlenmiştir. Aşağıdaki maddeler else if gibi ele alınmalıdır:

1. Eğer dönüştürme sırasında kaynak türün belirttiği değer hedef türün sınırları içerisinde kalıyorsa bilgi kaybı oluşmaz. Örneğin:

```
package csd;

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

		a = 100;

		b = (int)a; // Değer int sınırları içerisinde. Bilgi kaybı oluşmaz

		System.out.printf("b=%d\n", b);
	}
}
```

2. Büyük tamsayı türünden küçük tamsayı türüne dönüştürme yapıldığında sayının yüksek anlamlı byte değerleri atılır. Böylece sayı alakasız bir duruma gelebilir. Örneğin:

```
package csd;

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

		a = 1000000;

		b = (short)a; // Değer short sınırları içeirisinde değil bilgi kaybı oluşur

		System.out.printf("a=%X\n", a);
		System.out.printf("b=%X\n", b);
	}
}
```

Örneğin:

```
package csd;

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

		a = 0xABCDEFAA_FFFCAF1DL;

		b = (int)a; // Dikkat bilgi kaybı oluşur

		System.out.printf("a=%X\n", a);    // a=ABCDEFAAFFFCAF1D
```

```
		System.out.printf("b=%X\n", b);	// b=FFFCAF1D
	}
}
```

3. short türünden char türüne dönüştürme yapıldığında sayının işareti ne olursa olsun sayının bitleri işaretsiz olarak yorumlanır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		short a;
		char ch;

		a = 97;

		ch = (char)a;

		System.out.printf("ch=%c\n", ch); //ch=a
	}
}
```

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		short a;
		char ch;

		a = -97;

		ch = (char)a; // ch içerisinde 65439 var. En soldaki bit işaret biti olarak
yorumlanmadı

		System.out.printf("ch=%c\n", ch);
	}
}
```

4. char türünden short türüne dönüştürme yapıldığında sayının bit kalıbı değişmez yorumlanışı değişir. Yani işearet bitinin anlamı değişir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		char ch;
		short a;

		ch = '\uFFAB';
		a = (short) ch;

		System.out.printf("a=%d%n", a); //a=-85
		System.out.printf("ch=%c%n", ch);
	}
}
```

5. byte türünden char türüne tür dönüştürme operatörü ile yapılan dönüşüm iki aşamada gerçekleşir. Önce değer int türüne dönüştürülür. Sonra yüksek anlamlı byte değerleri atılarak char türüne dönüştürülür. Burada

bilindiği gibi byte türü içerisindeki değer negatifse sayının yüksek anlamlı bitleri 1(bir) ile beslenir, sayı pozitifse 0(sıfır) ile beslenir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		byte b;
		char ch;

		b = 97;
		ch = (char)b;

		System.out.printf("b=0x%X%n", b);
		System.out.printf("ch=%c%n", ch);
	}
}
```

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		byte b;
		char ch;

		b = -97;
		ch = (char)b;

		System.out.printf("b=0x%X%n", b);
		System.out.printf("ch=%c%n", ch);
	}
}
```

6. double türünden float türüne tür dönüştürme operatörü ile dönüştürme yapıldığında bilgi kaybı olabilir. Bu durumda IEEE 754 formatının yuvarlama kuralları uygulanarak float türünde gösterilebilecek en yakın sayı elde edilir. Örneğin:

```
package csd;

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

		a = 100.456789;

		b = (float)a;

		System.out.printf("a=%f\n", a); // a = 100,456789
		System.out.printf("b=%f\n", b); // b = 100,456787
	}
}
```

7. Gerçek sayı türünden tamsayı türüne tür dönüştürme operatörü ile yapılan dönüştürme işlemi iki aşamada gerçekleşir: Birinci aşamada hedef tür long türü dışında bir tamsayı türünde ise (byte, short, char veya int) önce int türüne dönüştürülür. Hedef tür long ise long türüne dönüştürülür. Bu dönüştürme sırasında sayının noktadan sonraki kısmı atılır. Bu durumda elde edilen değer dönüştürülecek türün (long veye int) sınırları içerisinde kalıyorsa bir sorun oluşmaz. Eğer elde edilen değer dönüştürülecek türün (long veya int) sınırları içerisinde kalmıyorsa ve sayı negatifse dönüştürülecek türün (long veya int) tutabildiği en küçük değer alınır, sayı pozitifse dönüştürülecek türün (long veya int) tutabildiği en büyük değer alınır. İkinci aşamada hedef tür int veya long ise değer birinci aşamada elde edilen değerdir. Hedef tür byte, short ya da char türlerinden biriyse elde edilen değer birinci aşamada elde değerin yüksek anlamlı byte değerleri atılmış biçimidir. Örneğin:

```
package csd;

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

		a = 100.45678;
		b = (int) a;

		System.out.printf("b=%d\n", b);
	}
}
```

Burada sayının noktadan sonraki kısmı atıldığında elde edilen değer int türünün sınırları içerisinde kaldığından doğrudan bu değer alınır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		double a = 5000000000.456;
		byte b = (byte)a;

		System.out.printf("a=%f%n", a);
		System.out.printf("b=0x%X%n", b);
		System.out.printf("b=%d%n", b);
	}
}
```

Burada dönüştürme byte türüne yapıldığından sayının noktadan sonraki kısmı atıldığında değer int türüne dönüştürülür. Sayının noktadan sonraki kısmı atıldığında elde edilen değer int türü sınırları dışındadır. Çok büyük bir değer olarak (pozitif olarak) sınırlar dışında kaldığından bu sayı int türü içerisinde tutulan en büyük sayıya (2147483647) dönüştürülür. Daha sonra sayının yüksek anlamlı 3(üç) byte'ı atılarak 1(bir) byte lık sayı elde edilir. Dikkat edilirse bu sayı -1 sayısıdır.

8. Tür dönüştürme operatörü ile hiçbir tür boolean türüne dönüştürülemez, boolean türü de hiçbir türe dönüştürülemez.

Bazen tür dönüştürme operatörü kullanılmazsa bilgi kaybı oluşabilir. Örneğin:

```
package csd;

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

		a = 2_000_000_000;
		b = 2;

		long result;

		result = a * b; //Dikkat: a * b işleminin sonucu int türünden olduğundan bilgi kaybı
olur

		System.out.printf("result=%d\n", result); //result=-294967296
	}
}
```

a * b işleminde a ve b her ikisi birden int türünden olduğundan sonuç int türünde çıkar, çarpımın sonucu int türünün sınırları dışında kaldığından yüksek anlamlı 4(dört) byte atılarak değer elde edilir. Burada bilgi kaybı olmaması için program aşağıdaki gibi yazılmalıdır:

```
package csd;

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

		a = 2_000_000_000;
		b = 2;

		long result;

		result = (long)a * b; // işlem long türünde yapılır bilgi kaybı olmaz

		System.out.printf("result=%d\n", result); // result=4000000000
	}
}
```

Dikkat edilirse a' nın içerisindeki değer bilinçli olarak (tür dönüştürme operatörü ile) long türüne dönüştürülmektedir. İşlem öncesinde b' nin içerisindeki değer de long türüne dönüştürülür. Sonuç long türünden çıkar ve bilgi kaybı oluşmaz.