

## Hafta 5: Ek Notlar

### Heap - Stack(Yığın)

Heap ve stack, programlama dillerinin çalışma zamanlarında kullanılan iki farklı bellek alanıdır.

<https://www.javatpoint.com/stack-vs-heap-java>

**Stack** ; metot yürütme sırasını ve yerel değişkenleri depolamak için kullanılır.

*Primitif tip dediğimiz **int**, **short**, **byte**, **long**, **decimal**, **double**, **float** gibi tipler value type (değer tipi) olarak adlandırılır ve stack de tutulur.*

- Stack, programda o anda çalışan, aktif olan metotlar için ayrılan bellek alanlarının olduğu yerdir. Bu sebeple **call stack** de denir.
- Her **metodun stackteki bellek alanına pencere (frame)** denir ve pencereler stackte üst üste açılırlar.
- Stack (yığın), “**last-in, first-out (LIFO)**” çalışır.

- Stackteki bellek pencereleri, kendisi için açıldığı metodun parametreleri, yerel değişkenleri vs. için bellek ihtiyaçlarını karşılar.
- Stack büyüklüğü, metodunun parametreleri ve o metotta tanımlanan değişkenler vb. bellek tüketen yapıların sayısı ve büyüklüğüne bağlıdır.
- Stack silinince, içindeki tüm veriler de silinir.

**Heap**; JVM tarafından nesne oluşturmak için kullanılan bellek alanıdır. Stackten çok daha büyüktür

- Nesneleri, JRE sınıflarını depolar. dinamik bellek ayırma ve bırakmayı yapar/kullanır.

```

StackDemo.java x
1 //created by ikok-27.03.2022
2 public class StackDemo {
3     public static void main(String[] args) {
4         System.out.printf("1: Main çalıştı\n");
5         f();
6         u( deger: 8);
7     }
8     public static void f(){
9         System.out.printf("2: F çalıştı\n");
10        g( b: false);
11        System.out.printf("4: g'den F'e dönüldü\n");
12        g( b: true);
13    }
14    public static void g(boolean b){
15        if(b==true){
16            System.out.printf("5: g()-> b=true - u tetikle\n");
17            u( deger: 6);
18            System.out.printf("7: u() sonrası g-f kapanış- \n");
19        }
20        else {
21            System.out.printf("3: g()-> b=False - f()'ye geri dön\n");}
22    }
23    public static void u(int deger){
24        System.out.printf("%d: U(%d) çalıştı\n",deger,deger);}
25    }

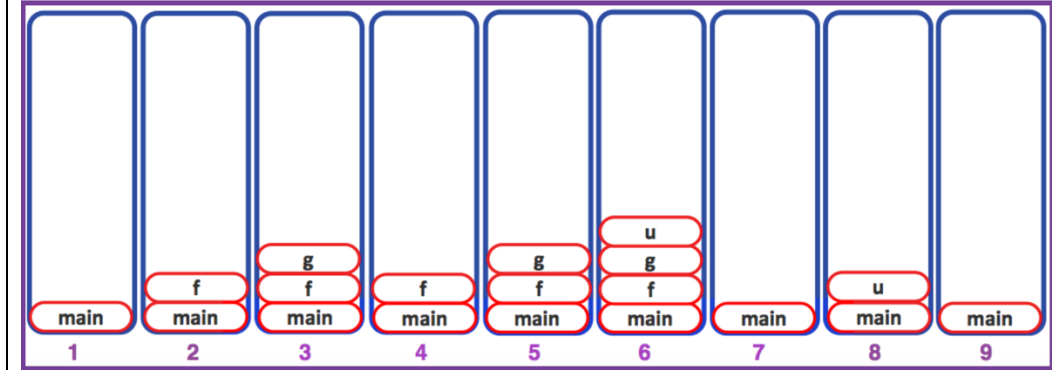
```

```

StackDemo x
↑ 1: Main çalıştı
↓ 2: F çalıştı
↺ 3: g()-> b=False - f()'ye geri dön
↻ 4: g'den F'e dönüldü
🖨 5: g()-> b=true - u tetikle
🗑 6: U(6) çalıştı
7: u() sonrası g-f kapanış-
8: U(8) çalıştı

```

Stack Demonun çalışmasıyla Stack'in değişimi



## Scope-Kapsam

Değişkenlerin erişilebildiği, görülebildiği alana, kapsam (scope) denir.

- Java'da kapsam, bloklar tarafından belirlenir.

### Değişkenlerin kapsamları, tanıtıldıkları yerden, içinde bulundukları bloğun sonuna kadardır.

- Üye değişkenlerin (member variables) kapsamı, tanıtıldıkları yerden sınıfın sonuna kadardır yani tüm sınıftır.
- Yerel değişkenlerin (local variables) kapsamı, tanıtıldıkları yerden, içinde bulundukları bloğun sonuna kadardır.
- Kapsama bloğun içindeki tüm alt kapsamlar da dahildir.
- Aynı kapsam içinde birden fazla aynı isimde, üye ya da yerel değişken olamaz.

- Fakat aynı kapsamda aynı isimde bir üye ve bir yerel değişken olabilir. • Bu durumda yerel değişken, üye değişkeni **gölgeler (shadowing)**.

- Java bu durumdaki bir değişkene erişimde, en yakında tanıtlana yani yerel değişkene erişir, üye değişkene ulaşmak için ayrı bir yapı kullanmak gerekir.

- Bir değişkene kapsamının dışında erişilemez.
  - Derleme hatası alınır.

Bu durumda o değişkenin erişilemediği, görülemediği, kapsam dışı (out of scope) olduğu söylenir.

## Çevrim(Conversion)-Casting

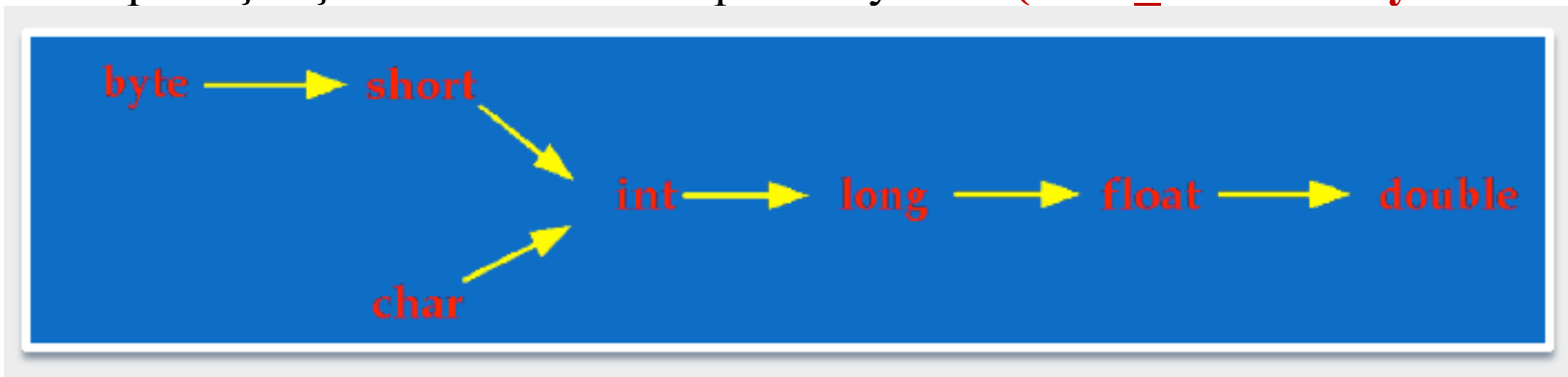
Bir değeri, sahip olduğu tipten farklı tipteki bir değişkene atamaya **çevrim (conversion)** denir.

- 1- Imkansız çevrim      Boolean b=12;  
(java: incompatible types: int cannot be converted to boolean)

### 2- Genişleten çevrim

```
int uzun =20;  
double d1=uzun;    // genişleten çevrim
```

- Java'da genişleten çevrimler otomatik olarak yapılır, çevrimin olması için atama yapmak dışında başka bir şeye gerek yoktur.
- Genişleten çevrimler hiçbir zaman çalışma zamanı hatası vermezler.
- İlkel tipler için çevrim-izin verilen promosyonlar **(NYP\_Hafta5-slayt no:35-Fig.6.4)**



### 3- Daraltan çevrim

Bit açısından daha geniş olan bir tipten daha dar olan bir tipe yapılan çevirimlerdir.

Daraltan çevirimler şunlardır:

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

Java'da daraltan çevirimler otomatik olarak yapılmaz, derleyici hata verir. Daraltan çevirim yapabilmek için **çevirme işlemcisi** (cast operator) olan **"()"** kullanılır:

- Atama yapılırken, çevirilen tip, çevirme işlemcisi içine yazılır.

```
double d2=12.45;
```

```
int i2= (int) d2; //daraltan çevrim-casting
```

Çevirme işlemcisi ile çalışma zamanında hata oluşmaz.

- Fakat çevirme **sonucunda bir veri kaybı olabilir.**

Çevirim sonucunda, çevirilen verinin büyüklüğüne bağlı olarak, **geniş tipteki bitlerin hepsi yeni tipe aktarılamayabilir.**