

Logikai kifejezések

A *logikai kifejezések* (*boolean expression*) olyan *kifejezések*, amelyekről a program egy pontján eldönthető, hogy igazak-e vagy hamisak. Azaz *típusa* logikai (*boolean*).

A matematikai józan parasztszre hagyatkozva ilyenek pl.

- `a < 5` (a kisebb, mint 5?)
- `c >= f` (c nagyobb vagy egyenlő f-fel?)
- `m + 2 > 7` (m+2 nagyobb, mint 7?)

Ha tudjuk az egyes változók értékét (a Java a program futtatása közben mindig tudni fogja), akkor mi is el tudjuk dönteni a program adott pontján, hogy ez a kifejezés igaz-e vagy hamis.

Az összehasonlító operátorokat (*comparision operator*) úgy írjuk le, ahogy kimondjuk: „nagyobb-egyenlő”, azaz `>=`.

Van két olyan összehasonlító operátor, amihez kell némi magyarázat:

- `==`, amit én csak „egyenlő-e” operátornak szoktam mondani (kettő egyenlőségjel), mert meg kell különböztetni az „értékadás” operátortól, ami **egy darab** = jel.

Azaz:

`a = 2`. Az a változó új értéke 2 lesz.

`a == 2`. Megvizsgáljuk, a értéke 2-e éppen.

- `!=`, nem-egyenlő. A „!” egyébként „nem”-et jelent.

A teljesség kedvéért az összehasonlító operátorok: `<`, `<=`, `>`, `>=`, `==`, `!=`.

Most már tudunk számokból (talán leggyakoribb változótípusunk) logikai kifejezést előállítani.

Ezeket az önállóan létező logikai kifejezéseket összekapcsolhatjuk a (logikai) ÉS, VAGY, NEM operátorokkal:

- `&&` - logikai ÉS. Akkor és csak akkor igaz az összekapcsolt logikai kifejezés, ha mindkét logikai kifejezés igaz.
- `||` - logikai VAGY. Akkor és csak akkor igaz az összekapcsolt logikai kifejezés, ha legalább az egyik logikai kifejezés igaz. (Persze mindkettő is lehet igaz).
- `!` – egy logikai kifejezés tagadása.

Példák:

- Az `a` változó 5 és 10 között van: `5 <= a && a <= 10`.
 - A józan paraszttéssel szemben a `5 <= a <= 10` **helytelen**.
Miért is?
A Java precedencia-táblázat 7. szintjén található a `<=` operátor, amit balról jobbra kell olvasni:
`5 <= a`-val kezdünk. Ennek értéke `a` értékének függvényében igaz vagy hamis (`true` vagy `false`). Ezután ezt a logikai értéket megpróbáljuk kapcsolatba hozni a másik felével: pl. `true <= a`. Ez pedig értelmetlen.
- `a == 2 || a == 4`. Az `a` értéke vagy 2 vagy 4.
- `!(a == 6 || a == 4)`. Az `a` változó értéke se nem 6, se nem 4. (Átírható `a != 6 && a != 4` alakra – ez az ún. DeMorgan-azonosság, bár elég, ha csak meg tudod csinálni, nem tudod a nevét)

Rövidzár kiértékelés (shortcut evaluation)

Abban az esetben, ha egy logikai kifejezés értéke `a` (balról jobbra) haladó kiértékelés során egy ponton bizonyossá válik, a Java abbahagyja az értelmezést, és továbblép az eredménnyel.

Tehát ha a fenti `a == 2 || a == 4` kifejezés esetén `a == 2` fennállásakor az `a == 4`-et már nem is nézi meg, hiszen tudjuk, hogy az eredmény `true`, ezen a további részkifejezések nem tudnak változtatni.

Ez nekünk jó, nem?

Általában igen, sőt, nagyon is:

```
nevező != 0 && számláló / nevező > 5
```

esetén tuti nem fogunk 0-val osztani, mert ha a `nevező` értéke 0, akkor az első fele hamis lesz, ami azt jelenti, hogy a második fél akármi is lesz, nem fog változtatni a teljes kifejezés összességében hamis értékén. Ennek következtében a Java azt le sem futtatja, így nem hajtja végre a `számláló / 0` osztást.

De oda is kell figyelni egy ilyen kifejezésre:

```
nevező == 0 || ++a > 3
```

esetén a `++a` csak akkor hajtódik végre, ha `nevező != 0`.

Ha van már némi előzetes tapasztalatod:

```
nevező == 0 || !kiirando(számláló, nevező)
```

esetén a kiirando() függvény (később tanuljuk), ami eldönti, hogy kiírjuk-e vagy sem a számot, egyáltalán nem fog elindulni, azaz csak akkor fut le, ha nevező $\neq 0$.