# 02. Tömbök 1.

## Tömbök

Az eddigiekben olyan változókat használtunk, amelyek egy értéket (egy számot, egy karaktert vagy egy stringet) tartalmaztak.

Ezzel szemben egy tömb több azonos típusú elemet is tartalmaz, például több számot vagy több stringet. Ezáltal lehetővé teszi, hogy sok adatot dolgozzunk fel.



A tömb típusú változókat is deklarálni kell a használat előtt. Például:

int[] szamok;  
double[] ertekek;

A szamok egy egész számokból álló tömb címét, az ertekek pedig egy törtekből álló tömb címét tartalmazhatja, de egyelőre még üres.

Ezután létre kell hozni a tömböket, és hozzárendelni őket a változókhoz (inicializálás):

szamok = new int[4];  
ertekek = new double[5+5];

Az első utasítás létrehoz egy 4 egész számból álló tömböt, és a címét eltárolja a szamok változóban.

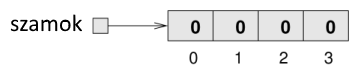
A második utasítás egy 10 törtszámból álló tömböt hoz létre, amelynek címe az ertekek változóba kerül.

A tömb méretét olyan kifejezéssel kell megadni, amelynek eredménye egy pozitív egész szám.

Ugyanúgy, mint az egyszerű változóknál, a deklaráció és az inicializálás egy sorban is történhet. Például így is csinálhattuk volna:

int[] szamok = new int[4];  
double[] ertekek = new double[5+5];

Egy egész számokból álló tömb létrehozásakor minden eleme 0 lesz. Az alábbi ábra mutatja a szamok tömböt a létrehozás után:



A nyíl azt jelzi, hogy a szamok változó csak a tömb címét tartalmazza. Az ilyen változókat nevezik referencia változónak. (Más objektumoknál, például a stringeknél is hasonló a helyzet. Egy String típusú változó csak a string címét tárolja, tehát szintén referencia változó.)

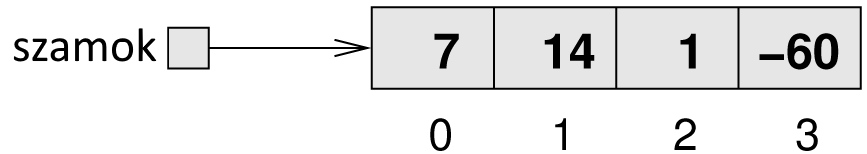
Az ábrán a téglalapokban lévő számok a tömb elemei, az alattuk lévő számok pedig a tömb indexei. Az indexek itt is 0-val kezdődnek, mint a stringeknél. A tömb egyes elemeire úgy hivatkozhatunk, hogy a tömb neve után szögletes zárójelek között megadjuk az indexet. Például a következő utasítás kiírja a tömb nulladik elemét:

System.out.println("A nulladik elem: " + szamok[0]);

A tömb elemei külön-külön is módosíthatók. Például:

szamok[0] = 7;  
szamok[1] = szamok[0] \*2;  
szamok[2]++;  
szamok[3] -= 60;

Nem kell sorban haladni, ezeknek az utasításoknak a sorrendje tetszőleges. Az eredményüket az alábbi ábra mutatja:

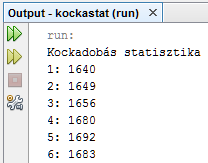


Ha egy tömb minden elemén szeretnénk végrehajtani utasításokat, akkor egy for ciklussal végigmegyünk a tömb elemein. Például, ha ki szeretnénk íratni a szamok tömb elemeit, akkor így csináljuk:

for (int i = 0; i < 4; i++)  
 System.out.println(szamok[i]);

## Statisztika

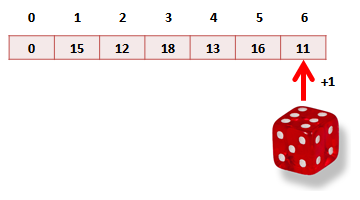
Készítsünk programot, amely 10 000-szer dob egy dobókockával, majd kiírja, hogy melyik értéket hányszor dobta! Minta:



Ha az egyes értékeket külön változókban tárolnánk (pl. az egyes nevű változó tartalmazná az egyes dobások számát, a kettes a kettesekét, …), akkor mindent 6-szor kellene elvégeznünk. Sokkal jobban járunk, ha egy tömböt alkalmazunk.

A tömbnek 6 eleműnek kellene lennie, de ott a nulladik elem is, bár azt nem szeretnénk használni. Ezért a tömbünk 7 elemű lesz, de a nulladik elemet figyelmen kívül hagyjuk.

A dobas tömb létrehozásakor 7 db 0 értéket fog tartalmazni, és minden dobás után hozzáadunk 1-et a megfelelő elemhez. Ha például 6-ost dobunk, akkor a 6-os indexűhöz adunk egyet. A program végén kiíratjuk a tömb elemeit.

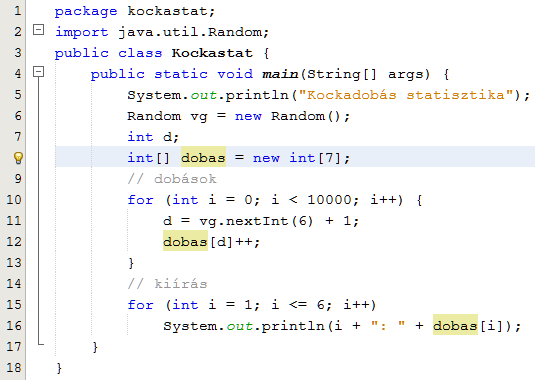


Mit fog tartalmazni a dobas[6] a program végén?   
Írd ide: azt, hogy hányszor dobtunk 6-ost

A program az inicializálások után két dolgot fog csinálni:

1. "Dob" 10 000-szer a kockával, és minden dobásnál megnöveli a megfelelő tömbelemet 1-gyel.
2. Végigmegy a tömb elemein, és kiírja őket.

Nézzük a programot:



A dobas tömböt a 8. sorban hozzuk létre. A 12. sorban megnöveljük a kisorsolt elemet eggyel. A 16. sorban pedig kiíratjuk az i-dik elemét (az 1-től kezdve!).

Készítsd el és próbáld ki a programot! Figyeld meg, hogy minden futásnál más eredményeket kapsz!

## Feladat

Gondold végig, hogyan lehetne tömb használata nélkül megoldani a feladatot!