# 3. Műveletek

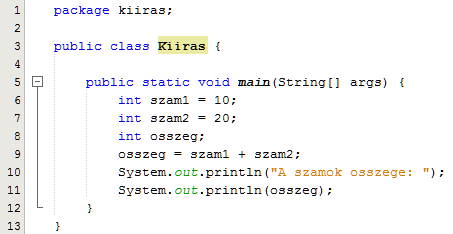
## Kiírás a képernyőre

A programozás során sokszor az a feladat, hogy valamilyen eredményt, vagy épp változók tartalmát írjuk ki a képernyőre.

System.out.println()  
System.out.print()

Kezdetben ezt a két metódust fogjuk kiíratásra használni. A Java nyelvben a szövegeket "idézőjelek" közé tesszük.

Amit idézőjelek közé írunk, azt a rendszer szövegnek (String-nek) tekinti. A kiíratás során a két metódusnak ilyen szövegeket szoktunk megadni, de ezekhez sokszor hozzá is fűzünk valamit. Lássunk erre példákat:



Az utolsó két sorban láthatod, hogy a println() metódusnak megadhatsz egy szöveget vagy egy változót is. A változót a kiíratás során átalakítja szöveggé, így a megjelenítés nem lesz gond.

Fontos, hogy az átalakítás csak a kiíratásra korlátozódik, az osszeg változó továbbra is azt az egész számot tartalmazza, amivel továbbra is végezhetsz számításokat.

Kezdj egy új projektet Kiiras néven, készítsd el és próbáld ki a programot!

Egy tipp a begépeléshez: A System.out.println() utasítást beviteléhez írd be, hogy *sout*, majd nyomd meg a Tab billentyűt!

A println() és print() metódusok között annyi a különbség, hogy a println() a kiírt szöveg után új sort kezd, a print() pedig a kurzort a sor végén hagyja. (ln = line = sor)

Ha az első println() metódust lecseréled print()-re, akkor a két kiírás egymás mellé kerül:



Próbáld ki!

Az utolsó két sort össze is vonhatod:

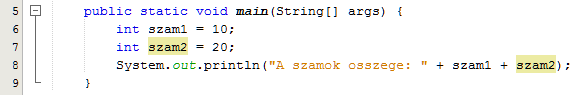


Ebben az esetben az történik, hogy az osszeg változó tartalmát, ami egy egész szám, hozzáfűzi a szöveghez úgy, hogy közben át is alakítja szöveg típusúvá.

Abban az esetben, amikor a + jel valamelyik oldalán egy szöveg található, akkor a + jel nem az összeadás, hanem az összefűzés műveletét jelenti!

Próbáld ki!

Tömörítsük még a programunkat, ne számítsuk ki külön változóba az összeget, hanem magába a kiíratásba tegyük bele az összeadást:

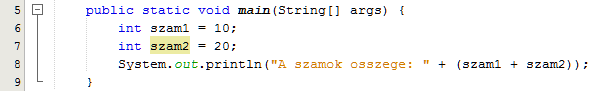


Futtasd le a módosított programot, majd nézd meg az eredményt!   
10+20=1020 ???

Mi a gond? A kifejezések kiértékelése során balról jobbra haladunk. Vagyis:

1. Kiírjuk a szöveget: "A számok összege: "
2. Hozzáfűzzük ehhez a szam1-et. "A számok összege: 10"
3. Hozzáfűzzük ehhez a szam2-őt: " A számok összege: 1020"

Mit tehetünk? Bíráljuk felül a műveleti sorrendet egy egyszerű zárójelezéssel:



Így már jó lesz az eredmény.

A kiíratás során a szövegben használhatunk olyan speciális vezérlő karaktereket, amelyek valamilyen plusz funkciót adnak hozzá a kiíratáshoz. Fontos, hogy ezeket a vezérlő karaktereket minden esetben \ jellel (backslash) vezetjük be, ami a mögötte elhelyezkedő karakternek más jelentést ad.

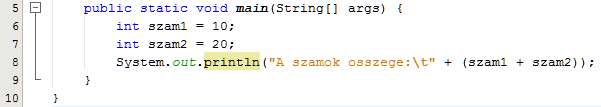
\n –új sor kezdés

\t – tabulátor

\\ – maga a \ karakter

\” – idézőjel

Példaképpen tegyünk egy tabulátor karaktert az összeg elé:



Így az összeg a szöveg után a következő tabulátor pozícióhoz kerül. Ez főleg akkor hasznos, ha az egymás alatti kiírásokat oszlopokba szeretnénk rendezni.

## Operátorok és operandusok

A programozási nyelvek fontos részét képezik az operátorok vagy műveleti jelek. A programozás során sokszor úgynevezett kifejezésekkel dolgozunk, amelyek valamilyen értékekből és közöttük értelmezett műveletekből állnak. Megnövelünk egy számot, összeadunk két változót, összehasonlítunk egy változót egy számmal, hogy egyenlőek-e, stb.

Operátornak magát a műveletet nevezzük, operandusnak pedig a kifejezés azon részét, amit változtatni akarunk vagy amit felhasználunk a számításhoz. Az operandus lehet egy megadott literál (egy konkrét érték), változó vagy kifejezés. Példák:

a = b;  
c = b + 2;  
i++;

Az előző példákban pirossal emeltem ki az operátorokat, a maradékok pedig az operandusok.

## Értékadó operátor

Ez az operátor hozzárendel egy értéket egy változóhoz. Az értékadás formája:

változó = kifejezés;

Az értékadás bal oldalán mindenképpen egy változónak kell szerepelnie, jobb oldalon pedig egy literál, változó, vagy olyan kifejezés (operátorok és operandusok összessége), amely egy értéket határoz meg, amit eltárolunk az értékadás bal oldalán lévő változóban. Például:

int a = 0;  
a = a + 1;

Az értékadó kifejezésben a jobb oldalon maga a bal oldali változó is szerepelhet. A második sorban egy olyan értékadás látható, ahol a kifejezésben is megtalálható a bal oldali változó, ebben a lépésben valójában a változót 1-gyel megnöveljük.

Fontos, hogy a kezdőérték megadásakor (inicializáció) ilyen nem lehetséges, mert addig nem használható fel egy változó egy kifejezésben, amíg nincs kezdőértéke! Itt az első sorban a kezdőérték megadása megtörtént, tehát utána már növelhetem ilyen értékadással.

## Inkrementáló operátorok

Létezik két speciális operátor, amely egy változó értékének 1-gyel való növelésére, illetve csökkentésére szolgál. Ezek használhatók egy változó előtt vagy után:

változó++;

++változó;

változó--;

--változó;

A ++ operátor megnöveli eggyel a változó értékét, a -- pedig csökkenti azt. Ezek a példák önálló utasításként működnek, ezért zártuk le őket ; jellel.

Amikor az operátor a változó mögött szerepel, azt postfix alaknak nevezzük, ha előtte, akkor prefix alakról beszélünk. Nyilván nem csak esztétikai jelentősége van, lássuk a gyakorlati hasznát!

Az előző példákban nincs különbség a két megoldás között. Azonban amikor a növelés vagy csökkentés egy kiíratás vagy összetettebb kifejezés része, akkor már fontos különbség adódik. Hasonlítsuk össze a következő két esetet:

int a = 10;  
System.out.println(a++); // 10  
System.out.println(a); // 11

int a = 10;  
System.out.println(++a); // 11  
System.out.println(a); // 11

Az első példában a növelés, mint művelet, a változó után található. Ez a gyakorlatban azt jelenti, hogy a kiíratás először felhasználja a változó eredeti értékét (10), majd ha minden művelet lezajlott ebben a sorban, akkor megnöveli a változó értékét (11). Vagyis a következő sorban, ahol már művelet nélkül írjuk ki, már a megnövelt értékét láthatjuk.

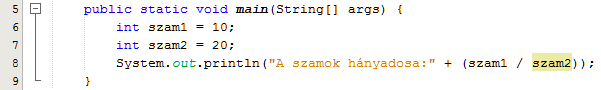
A második példában a növelés, mint művelet, a változó előtt található. Ez azt eredményezi, hogy ebben a sorban először megnöveli a változó értékét, majd a változó már megnövelt értékét írja ki. A következő sorban is ugyanazt az értéket írja ki, mivel itt szintén az előzőleg megnövelt értéket használhatjuk. Ugyanez igaz a csökkentésre is.

## Aritmetikai operátorok

Az operátorok közül leggyakrabban az aritmetikai operátorokat használjuk, amelyek matematikai műveleteket hajtanak végre két számértéken. Ezek az operátorok a következők: + – \* / %

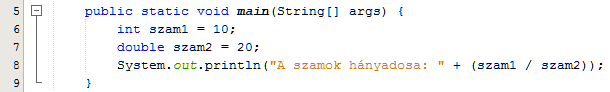
Az első hármat nem nagyon kell megmagyarázni, ezek matematikai alapműveletek, de az utolsó kettő már érdekesebb. A / jel, az osztás jele, másképp működik egész számoknál és másképp tört számoknál.

Amennyiben az osztást két egész szám között hajtjuk végre, akkor egész osztásról beszélünk. Ez azt adja meg, hogy hányszor van meg az egyik szám a másikban, és a maradékkal nem foglalkozunk. Írd át így a Kiiras programot:



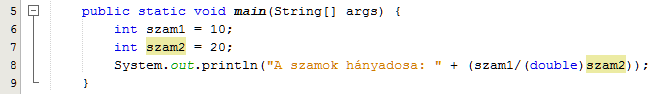
Az eredmény 0 lesz, mert 10-ben a 20 nullaszor van meg, és nem érdekel minket a maradék.

Az / operátor használatakor másfajta eredményt kapunk akkor, ha a két szám közül legalább az egyik nem egész (valós osztás). Írd át így a programot:



Az eredmény most 0.5 lesz.

Na de mi van akkor, ha két egész számunk van, de a teljes valós eredmény érdekel minket? Ilyenkor az osztásnál az egyik számot (ideiglenesen) átalakítjuk valóssá:



Mindegy, hogy a számlálót vagy a nevezőt alakítjuk át, az eredmény mindenképpen 0.5 lesz.

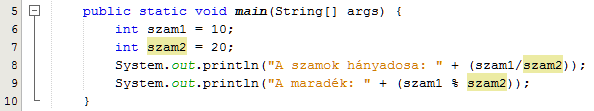
Ha nem változókat, hanem számokat (literálokat) osztunk, akkor az is elég, ha az egyik után odaírjuk a 0 törtrészt:

System.out.println(10/3.0);

vagy

System.out.println(10.0/3);

Amikor két szám osztásakor nem a hányados, hanem a maradék érdekel minket, akkor van szükségünk a % operátorra. Használni egyszerű:



Próbáld ki!

A maradékos osztást például számok osztóinak keresésekor szoktuk használni. Ha egy szám 5-tel osztva nulla maradékot ad, akkor osztható 5-tel.

A maradékos osztás egyébként nem csak egész számok között működik, akkor is helyes eredményt ad, ha nem egész mindkét szám. A lényeg, hogy ha az egyik szám nem egész, akkor az eredmény is a nem egész típusnak megfelelő lesz. Példák:

10 % 3  
10.0 % 3  
10 % 3.0  
10.0 % 3.0  
10.5 % 3  
10 % 3.5

Az értékadó operátor összekapcsolható valamilyen matematikai művelettel:

+=   
-=  
\*=  
/=  
%=

Ilyen operátor használatakor kiértékelésre kerül a jobb oldal, és a bal oldali változó értékét az értékadáshoz kapcsolt műveletnek megfelelően módosítja. Megnöveli a változó értékét a jobb oldallal, csökkenti a változó értékét a jobb oldallal, szorozza a változó értékét a jobb oldallal, osztja a változó értékét a jobb oldallal, stb.

Itt is igaz az, hogy ha a két oldalon egész értékek szerepelnek, akkor a /= egész osztást jelent, ha legalább az egyik valós érték, akkor valós osztás. Az utolsó művelet a bal oldali változó eredeti értékét osztja a jobb oldallal és a maradékot tárolja el a bal oldali változó új értékének.

Ha több művelet is van egy kifejezésben, akkor a Java először a szorzást, az osztást és a maradékképzést, utána pedig az összeadást és a kivonást végzi el. Azonos szintű műveletek esetén balról jobbra halad. A műveleti sorrendet zárójelekkel lehet módosítani.

## Feladatok

1. Kattints a Kiiras.java fül tetején lévő eszköztáron a History gombra! Ekkor egy listát kapsz a program előző változatairól. Az egyik időpontot kijelölve láthatod a módosításokat a mostani változathoz képest. A középen megjelenő nyilakkal tudod a korábbi változatot visszaállítani. A forráskódra a Source gombbal tudsz visszaváltani. Próbáld ki!

## Kérdések

1. Mi a következő kifejezések értéke, ha x egy egész típusú változó, és értéke 10?  
   a. x++ \* 2  
   b. ++x \* 2  
   c. (x+10) % 3  
   d. x + 10 % 3