# 2. Változók

## A változók

A programokban szinte mindig előfordul az, hogy valamilyen adatot tárolni szeretnénk. Erre szolgálnak a változók.

A változóknak alapvetően két fajtáját különböztetjük meg: a primitív és a referencia változókat.

A **primitív** változók egy egyszerű értéket tárolnak. Most ezeket változókat fogjuk tárgyalni.

A **referencia** típusú változók pedig egy objektumra mutatnak, maga a változó csak egy hivatkozás az objektum memóriabeli címére. Ezekről majd később lesz szó.

Minden változónak van típusa, neve és értéke.

A **típus** azt határozza meg, hogy milyen fajta értéket tárolhatunk az adott változóban.

A változók **neve** egyedi kell hogy legyen (egy blokkon belül). Tartalmazhat betűt, számjegyet, aláhúzást, dollár jelet. Nem kezdődhet számjeggyel. Kisbetűvel szoktuk kezdeni, de ha több szóból tesszük össze, a további szavak nagybetűvel kezdődnek (pl. alma, almaSulya). A névben lehet ékezetes betű is, de ritkán szoktak ékezetes nevet használni. A név nem lehet foglalt Java szó, például int, if, ...

A változók **értéke** alapvetően nem fix, ezt módosíthatjuk, erre utal a változó név. Minden változó egy időpontban egyetlen értéket tárolhat. Ha új értéket adunk neki, az előző érték törlődik. (Léteznek állandók is, melyek értékét csak egyszer adhatjuk meg.)

## Primitív típusok

A primitív változók alapvetően négyféle típusúak lehetnek, egy-egy konkrét értékkel (literállal):

* egész szám: 32
* valós szám: 1.125
* karakter: 'c'
* logikai érték: true

Ezek az alaptípusok, de az egész és valós típusok a bennük tárolt szám nagyságától függően még tovább bonthatók.

Az **egész** számok egész értékeket tárolnak tizedesjegyek nélkül.

A **valós** számok tizedesjegyekkel rendelkező számokat jelentenek.

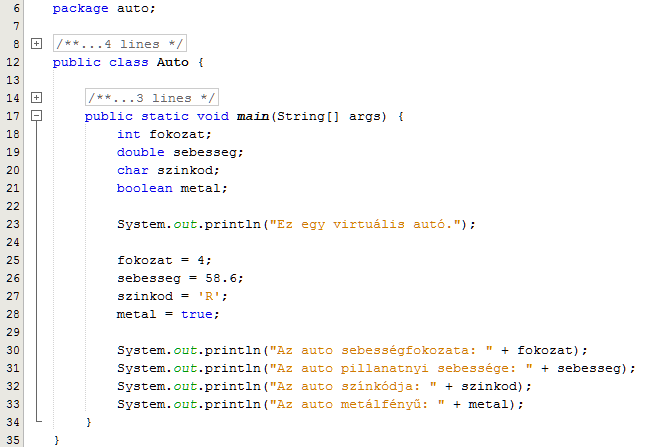
A **karakter** típusú változó valamilyen begépelhető karaktert tárolhat (betűt, számot, írásjelet, speciális karaktert, szóközt, stb).

A **logikai** érték pedig igaz vagy hamis értékeket tárolhat, pontosabban ezek angol nyelvű megfelelőjét (true, false).

## Példa

A változó tehát egy típussal, névvel és a típus által meghatározott értékkel rendelkező adat. Nézzünk a változók használatára egy példát!

Vegyünk egy Auto osztályt és pár ahhoz tartozó tulajdonságot. Legyen az autónak sebességfokozata, pillanatnyi sebessége, színkódja, és egy jelző, ami a színkódot bővíti ki azzal, hogy metálfényű-e vagy sem. Ebben a példában mind a négy alaptípus megtalálható:



A változók használatával kapcsolatban két nagyon fontos fogalmat kell tisztázni:

* deklarálás
* inicializálás

A **deklarálás** a változó típusának és nevének megadását jelenti. E nélkül a változó nem használható. Általános formája:

típus változónév;

Fontos! Mivel ezeket a változókat a main() metóduson belül deklaráltuk, ezek csak ebben a metódusban használhatók! (lokális változók)

Az **inicializálás** a változónak történő kezdőérték adás. Általános formája:

változónév = kezdőérték;

A deklarálás és az inicializálás akár össze is vonható, ekkor a következőt írjuk:

típus változónév = kezdőérték;

Ügyeljünk arra, hogy addig ne használjuk a változót, amíg nem rendelkezik kezdőértékkel! Használatnak minősül az is, ha a változó értékét ki szeretnénk íratni a képernyőre.

A változó értékét természetesen többször is meg lehet változtatni, ilyenkor az előző érték törlődik. Ez az **értékadás** formailag ugyanolyan, mint az inicializálás. A különbség a kettő között csak annyi, hogy az inicializálás a legelső értékadás.

A változó nevét használva a benne tárolt értéket kaphatjuk meg. Amikor például ki szeretnénk írni, hogy milyen értéket tárol, akkor a következőt írjuk:

"Az autó sebességfokozata: " + fokozat

Az idézőjelek közötti szövegrészt String-nek nevezzük. Ez egy karakterekből álló karakterlánc. A String egy referencia változótípus, később fogjuk tárgyalni, most elég annyit tudni róla, hogy amit idézőjelek közé teszünk, az String típusú lesz. Azért fontos ez, mert a System.out.println() kiírató metódus String-eket (szöveget) tud kiírni a képernyőre.

Az összeadás itt összefűzést jelent, vagyis a változó tartalmát odailleszti a szöveg végére. De a szöveg és a példában említett fokozat nevű változó nem egyforma típusú. Az összefűzés során a számérték szöveg típusúvá alakul egy automatikus **konverziót** (típusváltást) követően.

## További számtípusok

Az egész számokat a tárolható értékek alapján további típusokra osztjuk. Az, hogy a számítógép hány bájton tárolja a szám értékét, meghatározza azt az intervallumot, amekkora értéket a változó felvehet. Egész számokat a következő típusú változókban tárolhatunk:

|  |  |  |  |
| --- | --- | --- | --- |
| TÍPUS | LEÍRÁS | TÁROLÁS | INTERVALLUM |
| byte | bájt méretű egész | 8 bit | [-128;+127] |
| short | rövid egész | 16 bit | [-32768;+32767] |
| int | egész | 32 bit | [-2147483648;+2147483647] |
| long | hosszú egész | 64 bit | [-9223372036854775808; +9223372036854775807] |

A lebegőpontos számokat tartalmazó változóból is kétféle van:

|  |  |  |
| --- | --- | --- |
| TÍPUS | LEÍRÁS | TÁROLÁS |
| float | egyszeres pontosságú lebegőpontos | 32 bit |
| double | kétszeres pontosságú lebegőpontos | 64 bit |

A többféle egész vagy valós típus miatt már a kezdőérték megadásakor problémák lehetnek. Vegyük ezt a példát:

int szam = 10;

Ezzel semmi gond nincs, ugyanis az int típusba ez a méretű szám elfér. A byte és short típusoknál is hasonlóan lehet megadni kezdőértéket, arra kell csak ügyelni, hogy megfelelő méretű számot tároljunk csak benne. Az utolsó példánál már gond lenne, mert a tárolni kívánt érték már nincs benne a változónak megfelelő intervallumban:

byte b = 10;  
byte b = -40;  
byte b = 120;  
byte b = 130; // HIBÁS!

A short típussal is hasonló a helyzet, ott például már nem lehetne 35000-et tárolni, mert nem fér el ebben a típusú változóban.

Mi a helyzet a long típussal? Egy érdekes hibába futhatunk bele egy ilyen sorral:

long szam = 3000000000; // HIBÁS!

Tegyük fel, egy 3 milliárdos kezdőértéket akarunk adni. Fura, hiszen a változóban sokkal nagyobb méretű szám is beleférne. Ez a 3 milliárd melyik típusnak az intervallumából log ki? Az int típuséból. Arról van szó, hogy ha egy számot leírunk mindenféle sallang nélkül, azt a Java int típusú értékként kezeli. A kisebb számoknál akkor hogyan oldja meg, hogy egy byte változóba belerakhatja a nagyobb méretű int típusú értéket? Konverzióval, vagyis a byte változóban már az átalakított, megfelelő méretű értéket helyezi el. Igen ám, de maradva az előző hibánál itt a gond az, hogy ekkora méretű int szám nem is létezik, ezért el sem fogadja így leírva.

Ekkor külön jeleznünk kell a számérték végén a szám típusát a következőképp:

long szam = 3000000000L;

A leírt fix értékeket (nem csak ezeket, általában a leírt fix értékeket) **literálnak** nevezzük. Ez a long típus literálja. Érdekes módon a Java a 10L helyett a 10l literált is elfogadja, holott alaphelyzetben a kis és nagybetűk között különbséget tesz.

Nézzük, mi a helyzet, ha nem egy konkrét értéket, hanem egy másik változó értékét szertnénk egy változóban elhelyezni! A nagyobb méretű tárolókba elhelyezhetjük egy kisebb méretű változó értékét, de fordítva ez nem fordulhat elő, mert már fordítási hibával figyelmeztet a rendszer:

int i = 10;

short s = i; // HIBÁS!

Összefoglalva tehát az egész számokat:

* négyféle méretű típusa létezik (byte, short, int, long),
* nagyobb méretű típusba be lehet tenni a kisebbet, fordítva nem,
* a számokkal megadott literál int típusú, ha long típust szeretnél, akkor jelölni kell: 10L

A lebegőpontos számok esetén is hasonló a helyzet. Itt a jelzés nélküli literál (10.0) alaphelyzetben double típust jelent. Ha azonban mindenképpen float típust szeretnénk, akkor a 10F vagy 10f literál használatos. Itt is fordítási hibát ad, ha a nagyobb méretet a kisebb méretűben szeretnénk elhelyezni:

double d = 10.0;  
float f = d; // HIBÁS!

Tizedesvessző helyett tizedes pontot használunk, valamint nem kötelező ezt sem megadni ebben az esetben:

double d = 10D;  
float f = 10F;

Itt látszólag egész számot adunk kezdőértéknek, de a 10D literál miatt ez a 10.0 lesz valójában, vagy float típusnál hasonlóan csak kisebb méretben. A float típus értékmegadásánál viszont a literálnál kötelező ezt a formát használni, mivel az F elhagyásakor a számot alapból double méretűként kezelné a rendszer, ami viszont a float-ba nem fér bele. Ez a példa ezt a hibás használatot mutatja meg a jó megoldással egyűtt:

float f = 10.3; // HIBÁS! NEM KONVERTÁLJA!  
float f = 10.3F; // helyes

## Feladatok

1. Kezdj egy új projektet auto néven, és írd be a fenti programot! Próbáld ki!
2. Figyeld meg, hogy ha rákattintasz egy változó nevére, a NetBeans sárga háttérrel jelzi a változó többi előfordulását!

## Kérdések

1. Át tud-e alakítani a Java egy short típusú értéket long típusúvá? És fordítva?