1.4 Parancssori paraméterek, vezérlési szerkezetek

Parancssori paraméterek használata `javac`-vel való fordításnál:

```
java ProgramNeve param1 param2 param3...
```

Intellijben a paracssori paramétereket *Run/Debug Configuration* menüben a *Program arguments*nél tudod megadni.

Tömb bejárása a tömb `length` (hossz, vagyis elemszám) tulajdonságát felhasználva: (Természetesen nem csak a parancssori paraméterek tömbjének, hanem bármely tömbnek megnézhetjük az elemszámát a `length` tulajdonsággal.)

```
//parameterek bejarasa
for (int i=0; i<args.length; i++) {
    System.out.println("" + (i + 1) + ". parameter: " + args[i]);
    //szam szovegge alakitasa
    String str = "" + i;
}
```

Paraméterek összege

A main függvény egy sztring tömböt kap, ezt nem lehet egyszeren felülírni. Ha számként akarjuk kezelni a kapott paramétereket, akkor az `Integer.parseInt()` függvényt kell használnunk.

```
//parameterek osszege
int osszeg = 0;
for (int i = 0; i < args.length; i++) {
    osszeg += Integer.parseInt(args[i]);
}
System.out.println("A parameterek osszege: " + osszeg);
```

Adattípusok

Egyszer (primitív) adattípusok: **boolean, char, byte, short, int, long, float, double**, ezek nagy része ismers lehet C-bl. A C-vel ellentétben Java-ban létezik egy beépített primitív típus logikai érték tárolására, amelyet `boolean` típusnak nevezünk, és értéke csak `true` vagy `false`lehet. További eltérés a C-hez képest, hogy nincs eljeltelen típus, tehát nem használhatjuk az `unsigned` kulcsszót, csak és kizárólag eljeles típusokat hozhatunk létre. Bvebben ezen a linken olvashatsz a primitív adattípusokról. Egy érdekes cikk a lebegpontos számokról, számábrázolásról.

> ***Megjegyzés**: Ha valami miatt azonban mégis szükség lenne egy eljeltelen egészre, akkor Java 8 (vagy afeletti verzió) esetén használhatjuk az `Integer` osztály néhány erre a célra létrehozott metódusát, mint például a compareUnsigned, divideUnsigned.*

Később látni fogjuk ennek hasznát, de alapveten minden Java-beli primitív típusnak létezik egy csomagoló osztálya, amellyel egy primitív típusú adatból objektumot készíthetünk, "becsomagolhatjuk" azt. A csomagoló (wrapper) osztályok a következk (sorrendjük megegyezik a primitív típusoknál történt felsorolás sorrendjével): **Boolean, Character, Integer, Long, Float, Double**. Egy összefoglaló táblázat a beépített típusokról:

Típus neve	Érték	Default érték	Méret	Értéktartomány
boolean	1 bitet reprezentál	false	nincs precíz definíció	true/false
char	16 bites unicode karakter	\u0000	2 byte	0 - 65535
byte	8 bites egész	0	1 byte	-128 - 127
short	16 bites egész	0	2 byte	-32768 - 32767
int	32 bites egész	0	4 byte	-2147483648 - 2147483647
long	64 bites egész	0L	8 byte	-9223372036854775808 -,9223372036854775807
float	32 bites lebegpontos (IEEE 754)	0.0f	4 byte	1.40129846432481707e-45 - 3.40282346638528860e38 (pozitív vagy negatív), +/- végtelen, +/- 0, NaN
double	64 bites lebegpontos (IEEE 754)	0.0d	8 byte	4.94065645841246544e-324d - 1.79769313486231570e+308d (pozitív vagy negatív), +/- végtelen, +/- 0, NaN

A beépített típusokat bemutató példakód elérhet a `pub/Programozas-I/nappali/gyakorlat/03/PrimitivTipusok.java` útvonalon.

```java
public class PrimitivTipusok {

    public static void main(String[] args) {
    boolean bo = true; // logikai tipus
    char c1 = 'a'; // karakter tipus
    char c2 = '\u0054'; // ket bajtos, unicode karaktereket tarol!
    byte b1 = 127; // 8 bites egesz tipus
    byte b2 = -128; // minden egesz tipus elojeles!
    short s = 1024; // 16 bites egesz
    int i = 0x7fffffff; // 32 bites egesz
    long l = 0x7fffffffffffffffL; // 64 bites egesz
    float f = 123.123f; // 32 bites lebegopontos tipus
    double d = 5.0; // 64 bites lebegopontos

    // kiiras konzolra
    System.out.println(bo);
    System.out.println(c1);
    System.out.println(c2);
    System.out.println(b1);
    System.out.println(b2);
    System.out.println(s);
    System.out.println(i);
    System.out.println(l);
    System.out.println(f);
    System.out.println(d);
    }
}
```

A lebegpontos számokkal azonban óvatosan kell bánni. Erre egy tökéletes példa a Süti program. A történet a következ: ellátogatunk az Egyesült Államokba, de sajnos hamarosan indulunk is tovább, így csak a reptéri cukrászdában vásárolhatunk sütit. A sietségünket azonban kihasználja a reptéri cukrász: elcsábít minket a konkurencia ell, a 0.1 dolláros sütivel, azonban minden egyes következ sütiért 0.1 dollárral többet kér, mint amennyibe az elz került. Vajon hány sütit ehetünk a maradék 1 dollárunkból? Írjunk rá egy programot, menstük el `Suti.java`néven.

```java
public class Suti {

    public static void main(String[] args) {
        double penzunk = 1.00;
        int megvettSutik = 0;
        for (double ar = 0.1; penzunk >= ar; ar += 0.1) {
            penzunk -= ar;
            ++megvettSutik;
        }
        System.out.println("Megvett sütik: " + megvettSutik);
        System.out.println("Megmaradt pénz: " + penzunk);
    }
}
```

Hm.. valami nem stimmel. Számoljuk ki kézzel is, egy egyszer táblázat segítségével:

Pénzünk	Következ süti ára	Megevett sütik száma
1.0 $	0.1 $	0
0.9 $	0.2 $	1
0.7 $	0.3 $	2
0.4 $	0.4 $	3
0.0 $	lényegtelen	**4**

A fenti példa által láthatjuk, hogy pl.: valuta tárolásához sosem érdemes `float` vagy `double`típust választani. Bvebben errl a problémáról ezen é s ezen linken.

String (szöveges) típus

String típus. A sztringeket az eddigiekkel szemben itt már szeretjük, kezelésük könny. Ezt a típust **mindig** nagy kezdbetvel írjuk.

Operátorok

`+, -, *, /, %` operátorok és használatuk ugyanaz, mint C-ben.

A + jel sztringekre konkatenációt (összefzést) jelöl. Használata nagyon intuitív. Például `"kutya" + "mutya" -> "kutyamutya"`

A C-beli pointereknél használt * (dereferencia) és & (címképzés) operátor itt **nem** létezik.

>> - aritmetikai eltolás: megrzi az eljelbitet.

>>> - logikai eltolás: az eljelbitet is tolja. (Tehát nem rzi meg a szám negativitását)

Az operátorokról bvebben olvashatsz az alábbi linkeken: Summary of Operators, Java Operators Tutorial.

Vezérlési szerkezetek

Majdnem minden mködik, ami C-ben is mködött, szóval elvileg aki idáig eljutott, ezeket már ismeri. A reláció operátorok is ugyanúgy mködnek, mint C-ben, illetve a ++ és `--` operátorok is ugyanúgy használhatóak.

Létezik tehát a már megszokott módon az `if()` és a `?:` operátor is. Az if komplexebb feltételek megadására is alkalmas. Például ha azt szeretnénk, hogy egy x változó 3 és 7 között legyen, vagy 13, vagy negatív és nem -9, akkor ezt a következ kóddal adhatjuk meg:

```
if ((x >= 3 && x <= 7) || x == 13 || (x < 0 && x != 9)) {
    System.out.println("A feltétel teljesül.");
}
```

Logikai kifejezéseket tehát összefzhetünk a && (és) és a || (vagy) operátorok használatával. A zárójelezés ilyenkor kritikus fontosságú lehet.

A boolean típus kezelhet logikai kifejezésként, mivel pontosan az is, azaz egy igaz vagy hamis értéket tárol. Ilyenkor operátorok nélkül is igazságértéket jelképeznek.

```
boolean pozitiv = true;
if (pozitiv) System.out.println("A szám pozitív.");
if (!pozitív) System.out.println("A szám nem pozitív.");
```

A `?:` operátor használata például a kiíráson belül lehet indokolt. Az elz példát például jelentsen egyszersíti:

```
boolean pozitiv = true;
System.out.println("A szám " + ((pozitív) ? "" : "nem ") + "pozitív.");
```

Viszont, C-vel ellentétben itt nincs implicit konverzió `int` és `boolean` között, a Java megköveteli az összehasonlító operátorok használatát ebben az esetben:

```
int x = 1;
if (x) { //Javaban ez így nem mködik!
  System.out.println("Minden rendben!");
}
//A helyes feltétel az alábbi:
if (x > 0) {
  System.out.println("Minden rendben!");
}
```

A `while`, `do-while` és `switch` utasítás megegyezik a C-ben tanultakkal. A for utasításnál viszont itt kényelmesebben deklarálható ciklusváltozó, mivel itt az utasításon belül is megtehetjük ezt (viszont ezesetben a cikluson kívül már **nem** látjuk a ciklusváltozót!). Tehát a C-vel ellentétben itt már mködne a következ példa:

```
for (int i = 0; i < 5; i++) {
    //ciklus tartalma
}
```

Az egyetlen dolog, amely nem mködik Java-ban, de a C-ben igen, az a `goto` utasítás. A program egyes részeit ugyanúgy címkézhetjük, azonban a `goto` parancsot nem használhatjuk. Ez Javaban ugyan ez foglalt kulcsszó, de implementáció nem tartozik hozzá. Ennek oka az, hogy a nyelv készíti kulcsszóvá tették a `goto`-t, azonban eddig még nem jött tömeges nyomás, nem volt olyan probléma, amelyet ezen kulcsszó nélkül ne tudtak volna megoldani, így egyelre ez egy fenntartott kulcsszó (azaz például ilyen nev változót nem hozhatunk létre), de semmiféle mködése nincs.

Memóriaterületek

A következekben röviden bemutatjuk a stacket, illetve a heapet, azonban ezek a témakörök az eladáson ennél jóval részletesebben be lettek mutatva. Bvebben olvashatunk a témáról a következ linkeken: JVM specifikáció ide vonatkozó részei

Stack

A stack-en tároljuk a primitív típusú adatokat. Ez a memóriaterület nagyon gyors elérés, általában lokális változókat, rövid élet adatokat tárolunk benne. A stacken létrehozott változók élettartalma a létrehozó blokkra korlátozódik, azaz pl.: ha van egy lokális változónk egy függvényben, akkor az a változó addig él, és addig lesz a memóriában tárolva, ameddig a függvény futása be nem fejezdik. Ezt követen a foglalt memóriaterület automatikusan felszabadul. Hátránya a heap memóriával szemben az, hogy a stack jóval kisebb. Javaban csak és kizárólag primitív típusú adatokat tárolhatunk a stacken (szám, bájt, karakter vagy logikai érték).

A gyakorlatban, ha egy változó értékét stacken tároljuk (pl.: `int x = 42;`), akkor akármikor hivatkozunk a változóra, a változó neve gyakorlatilag egyet jelent az értékkel, érték szerint tárolódik.

Heap

A heap memórián tároljuk az objektumokat, tömböket (ezeket referencia típusúaknak hívjuk). Ez a stack-kel ellentétben egy lassabb, dinamikus futás-közbeni memóriaterület, cserébe viszont jóval nagyobb terület áll a programunk rendelkezésére. Az itt létrehozott adatokat hosszabb távon tároljuk, globálisan léteznek, nem csak egy-egy függvény belsejében, lokálisan.

Amikor létrehozunk egy objektumot a heapen, akkor az adott változó az gyakorlatilag csak egy referencia, mutató lesz a heap-beli memóriaterületre, ahol tényleges az objektum tárolódik, a változó maga csak ezt a referenciát tárolja. *Ez ismers lehet, a pointer fogalma hasonló.* Tehát a referencia típusú változó nem közvetlenül tárolja az értéket, csak egy referenciát tárol, és a heapen tárolt értéket közvetetten, ezen a referencián keresztül érjük el.

> ***Gyakorlati példa**: Egy nagyon szemléletes példa, Bruce Eckel Thinking in Java cím könyvébl: Képzeljük el, hogy van egy `Televízió` típusú objektumunk a heapen, melyhez tartozik egy referencia, amelyet letárolunk, és ennek a neve legyen `távirányító`. Ahogy a való életben is, ülünk a kényelmes fotelünkben, és nézzük a televíziót, de amikor szeretnénk a `Televízió` objektumunkon bármely módosítást eszközölni (halkítani, hangosítani, csatornát váltani, be-, kikapcsolni), akkor a `távirányító` segítségével tesszük meg ezt. És ha a szobában sétálva, lefekve bárhol szeretnénk a `Televízió` objektumon machinálni, akkor a távirányítót visszük magunkkal, nem pedig magát a `Televíziót`.*

Alapértelmezett kimenetek

Ahogy már beszéltünk róla, a Java nyelvben szöveget írhatunk ki a képernyre a `System.out.println()` paranccsal. Ez viszont nem feltétlenül mindig a terminált jelenti. A `System.out` az alapértelmezett kimenet, ami esetünkben a terminálra volt irányítva. Ezt át is lehet állítani, hogy máshova írjunk vele, például egy fájlba.

Az eddig tanult kiíratáshoz hasonló az alapértelmezett (vagy default) hibakimenet, melynek fogalma szintén ismers lehet. Erre a `System.err.println()` paranccsal tudunk írni a már látott módon. Ez hibaüzenetek közlésére szolgál. Egyetlen szemmel látható különbsége az alapértelmezett kimenettl, hogy az Intelllj konzolján a szöveg pirossal jelenik meg.

A két kimenet viszont lehet két különböz helyre is irányítva.

A kimenetek mködése aszinkron. Ez azt jelenti, hogy nem feltétlenül pont akkor írja ki a dolgokat, amikor mi utasítjuk rá, ha a kimenet éppen nem tudja ellátni a feladatát, akkor a kiírandó adat várakozik. Tehát a következ kódnak két lehetséges kimenetele lehet:

```
System.out.println("Most hibaüzenet következik:");
System.err.println("Ez a hibaüzenet.");
```

Az egyik kimenetel:

```
Most hibaüzenet következik:
Ez a hibaüzenet.
```

A másik pedig:

```
Ez a hibaüzenet.
Most hibaüzenet következik:
```

Ebbl következik, hogy a hibakimenet **NEM** használható a szöveg egyszer színezésére, mert összekavarodhat a mondanivalónk. Attól persze nem kell félni, hogy a `System.out`-ra küldött két üzenet összekeveredik, ezek sorrendben várakoznak.

A gyorsabb munka érdekében az Intelllj megkönnyíti a kimenetre írást. A default kimenet eléréséhez egyszeren beírhatjuk a `sout` szót, majd Ctrl+Space hatására kiegészíthetjük `System.out.println()`-né.

Feladatok

1. Írd ki a parancssori paramétereket a konzolra
2. Írd ki a parancssori paramétereket a konzolra, fordított sorrendben

3. Írd ki a parancssori paramétereket úgy, hogy az n. sorban az els n darab parancssori paramétert írja ki: els sorba csak az elst, a másodikba az els kettt szóközzel elválasztva, a harmadikba az els hármat, stb..

```
java Main egy ketto kutya cica fagyi
```

```
egy
egy ketto
egy ketto kutya
egy ketto kutya cica
egy ketto kutya cica fagyi
```

4. Írd ki a parancssori paraméterek közül a legnagyobbat, legkisebbet, valamint az értékek átlagát.
5. A parancssori paraméterek alapján döntsd el, hogy egy a bemen számok számtani, mértani sorozatot alkotnak-e, vagy esetleg egyiket sem. Feltehetjük, hogy mindegyik egész szám, és legalább 3 db paraméterünk van. Az összegképletek:
    - számtani: **an = a1 + (n – 1) * d**
    - mértani: **an = a1 * q n – 1**
6. Számítsd ki a parancssoron kapott két idpont (óra perc óra perc) között eltelt idt, és írasd ki a konzolra (óra perc formában). A program elkészítése során ügyelj az adatok helyességére
    - bemen paraméterek száma
    - az órák 0-23 intervallumba kell, hogy essenek
    - a percek 0-59 intervallumba kell, hogy essenek