Programozási nyelvek Java

2. gyakorlat

Függvények

Általános prototípus

```
<módosítószavak> <visszatérési érték> <név>( <paraméterek listája> )
[ throws <kivétel lista> ] {

}
```

Módosítószavak

- Láthatóság: public, protected, private. Ha nem definiált, akkor úgynevezett package-private láthatóság.
- Lehet abstract - leszármazottban kötelezően felüldefiniálandó
- Lehet nal - felüldefiniálhatóság letiltására
- Lehet static - osztály szintű függvény (megjegyzés: static kontextusból csak static módosítóval ellátott hivatkozás szerepelhet)
- Egyéb, pl. strictfp, native, synchronized, transient, volatile (utóbbi kettő csak field-ekre). Ezekről később.

Visszatérési érték szerinti csoportosítás

- void esetében eljárás
- Minden egyéb esetben függvényről beszélünk

Szignatúra, paraméterátadás

- Szignatúra: A függvény neve és paramétereinek típusa.
  Például: eredmenyMeghatarozasa(double, int, int)
- Minden paraméter **érték** szerint adódik át, azaz a paraméter helyén szereplő értékről másolat készül. A metódus ezt a másolatot látja, használja. Persze, ha a másolat módosul, az az eredeti értéket nem érinti.

DE, amennyiben a paraméter nem valamelyik beépített, egyszerű típusba tartozik, úgy az átadásnál a referenciáról készül másolat, azaz a C++ fogalmai szerint ilyenkor referencia szerinti átadás történik, tehát a hivatkozott objektum a függvény belsejében megváltoztatható.

További típusok

Tömbök

A tömb adott számú, azonos típusú elemeket tartalmazó adattípus. A tömb indexe 0-val kezdődik. A tömbök egynél több értéket is tárolhatnak, de az elemek száma rögzített.

- Minden T típushoz létezik tömb. (T[])
- A **new** operátorral dinamikusan lehet létrehozni
- Inicializálásnál az 1. dimenzió megadása kötelező
  int arr2[];
  int[] arr1 = new int[5];
  int arr3[] = { 1,2,3,4,5 }; -> ekkor a fordító hozza létre
- Elemeinek lekérdezése: arr3[0], arr1[i]
- Többdimenziós tömbök inicializálásánál az 1. dimenziót kell csak megadni, de ha tudod a többit is, akkor azokat is érdemes: int[][] arr = new int[9][9];
- A tömbök mérete a **length** adatmezővel kérdezhető le
- Túlindexelés esetén kivétel keletkezik.

Érdekesség, hogy nem csak "négyszögletes" kétdimenziós tömböt lehet készíteni, hanem olyat is, ahol az egyes sorok nem azonos hosszúak.

String típus

A String típus char típusú elemek tömbje.

A String típusú objektumok megváltoztathatatlanok (immutable). Amikor módosítunk egy stringet, akkor automatikusan egy új példány jön létre a memóriában, ez pedig nem feltétlenül „olcsó” művelet. Ha többször van szükségünk erre, akkor használjunk inkább a StringBuilder-t.

Műveletek

- http://download.oracle.com/javase/6/docs/api/java/lang/String.html
- A **substring** függvény esetében, amely az eredeti szöveg egy szeletét adja vissza. Ekkor nem keletkezik új tömb, hanem csak az eredeti tömbön beállítja a kezdő- és a végreferenciát.
- indexOf, lastIndexOf, startsWith, length, …

Szövegkonverzió

- Stringgé: String s = "" + 1; (precedenciára figyelni!) vagy a toString() függvény
- Stringből: Integer.parseInt("1"), Double.parseDouble("2"), ...

Stringek összehasonlítása

- Mint az objektumokat: equals() metódussal
- az == operátor referencia szerinti összehasonlítást végez csak, nem tartalom szerint
- "a".equals("a") - érték szerinti összehasonlítás

StringBuilder

- http://download.oracle.com/javase/1.5.0/docs/api/java/lang/StringBuilder.html
- http://download.oracle.com/javase/tutorial/java/data/buffers.html
- Ha többször van szükségünk egy String objektum módosítására, akkor használjunk inkább a StringBuilder-t.

Automatikusan lefoglal egy nagyobb darab memóriát, és ha ez betelik, akkor allokál egy nagyobb méretű területet és átmásolja magát oda.

Parancssori argumentumok

- Minden Java programnak adhatunk indításkor paraméterek, amiket a program egy tömbben tárol
- public static void main(**String[] args**)
  Az args tömb tárolja a parancssori paramétereket String formátumban, amelyeket a tömb megfelelő indexelésével érhetünk el
- C, C++-szal ellentétben a 0-dik indexen lévő argumentum nem a program neve!
- Paramétereknek a darabszámát könnyedén megállapíthatjuk:
  args.length, ami egy int értéket ad vissza
- Ha számokkal akarunk dolgozni, akkor az argumentumokat át kell alakítani
  (Ez csak akkor fog működni, ha eleve számokat adtunk meg paramétereknek, különben kivétel keletkezik)
- Átalakításra példa:
  int i = Integer.parseInt(”12”);
  int i = Integer.parseInt(args[0]);
  double d = Double.parseDouble(args[2]);

Konzolos adatbevitel

Scanner

Scanner osztálybeli objektummal valósítható meg a konzolos beolvasás.

- java.util csomagban található
- Konstruktorában megadható **StreamInput** és **File** típusú objektum is
    - Konzolról való olvasás esetén: System.in
    - Fileból olvasás esetén: new File(<fájlnév>)
        - java.io csomagban található
- **close()**: scanner lezárása
- **nextBoolean()**: beolvassa és átkonvertálja az értéket
- **nextInt()**: beolvassa és átkonvertálja az értéket
- **nextLong()**: beolvassa és átkonvertálja az értéket
- **nextDouble()**: beolvassa és átkonvertálja az értéket
- **next()**: beolvassa és átkonvertálja az értéket
- **nextLine()**: olvas a következő új sorig, és visszatér egy String típussal (beolvasott szöveg)
- **hasNextBoolean()**: igaz, ha a következő beolvasandó érték boolean típusú
- **hasNextInt()**: igaz, ha a következő beolvasandó érték int típusú
- **hasNextLong()**: igaz, ha a következő beolvasandó érték long típusú
- **hasNextDouble()**: igaz, ha a következő beolvasandó érték double típusú
- **hasNext()**: igaz, van még mit olvasni
- **hasNextLine()**: igaz, ha van még egy input sor

Példa

```
Scanner in = new Scanner(System.in);

if(in.hasNextInt())
{ System.out.println(in.nextInt()); }

in.close();
```

Ez a kis kódrészlet vár egy inputot a konzolból. Ha egész számot adunk meg, akkor kiírja, különben tovább ugrik a vezérlés.

Feladatok

1. Írj egy ciklust, ami addig olvas a konzolról, amíg egész számot kap. Ezután írja ki a beolvasott számok összegét.
2. Írd át az előző feladatot úgy, h egy test.txt nevű fájlból olvasson. (A fájl a honlapon megtalálható!)
3. Készíts egy függvényt, ami bementként kap egy int típusú elemekből álló tömböt, és végrehajt rajta egy maximumkeresést, majd visszatér az maximum elem indexével. Végül az elem indexét írjuk ki a képernyőre.
4. Készíts a 3. feladathoz egy konzolos menüt!
   A menüben legyen 2 opció:
   1 – Konzolról olvasás
   2 – Fájlból olvasás
   1) Konzolos beolvasás esetén először kérjük be a tömb hosszát, majd kérjünk be annyi adatot, amennyi a tömb hossza. A felhasználónak természetesen jelezzük, hogy éppen melyik adatot várjuk.
   2) A fájlból olvasás esetén kérje be a fájlnevet, majd olvassa be azt.
      A fájl első sora a tömb hosszát jelzi, a többi pedig a tömb elemei.
      Használd a test.txt fájlt.

   Miután meghatározta a maximális indexet, írja ki a képernyőre, majd kérdezze meg a program, hogy kívánja-e újrafuttatni az alkalmazást. Újrafuttatás esetén jelenítsük meg ismét a főmenüt.

5. Valósítsd meg a mátrixösszeadást!

```
int[][] MatrixAddition(int[][] m1, int[][] m2)
```

   A programnak legyen egy konzolos menüje (hasonlóan a 4. feladathoz).
   Legelőször be kell kérni egy pozitív egész számot, amely mindkét mátrix sorainak és oszlopainak a száma lesz. A mátrixok négyzetesek (N x N). Ezután be kell kérni a két mátrix adatait. Majd végezd el a mátrixok összeadását, és írasd ki konzolra ez elemeit (mátrix alakban).
   Végül pedig kérdezze meg a program a **konzolban**, hogy kívánom-e újrafuttatni a programot.