Home

Last edited by **javagyak** a week ago

Clone repository (/javagyak/03/wikis/git_access)

Home (/javagyak/03/wikis/home)

Java SE 8 Documentation (http://docs.oracle.com/javase/8/docs/) Feladatok

Tömbök használata

Tömbök segítségével azonos típusú elemekből tudunk felépíteni olyan adatszerkezetet, ahol az egyes elemeket egy index vagy kulcs segítségével lehet azonosítani és hivatkozni. Mivel az elemek típusa azonos és így mindegyikük egyenlő méretű helyet foglal, valamint mivel a tömb elemei folytonosan helyezkednek el a memóriában, az elemek helyét a memóriában egy megfelelő képlet segítségével tudjuk kiszámítani.

Például egydimenziós esetben:

```
a[i] = a + i * sizeof(T)
```

ahol `a` tömb, az `[i]` az indexelő operátor, amelyben `i` az elérendő elem indexét (lényegében sorszámát) jelenti. A `sizeof()` operátor egy típus értékeinek méretét adja meg byte-ban, végül a `T` tömb elemeinek típusa. A tömb nevének hivatkozása önmaga a tömb kezdőcímét adja meg a memóriában. Ahogy a fenti képletből is sejthető, az indexelést nullától kezdjük.

A Java nyelvben a tömbök egy olyan speciális objektumtípushoz tartoznak, amelyek szintaxisa különbözik az objektumok hagyományos szintaxisától.

- Minden `T` típushoz tudunk rendelni egy `T[]` típust, amely lényegében a `T` elemekből képzett tömböt jelenti. Például:

  ```
  int[] intArray;
  char[] charArray;
  String[] stringArray;
  ```

- A tömböket az objektumokhoz hasonló módon példányosítani kell. Ennek elmulasztásával az értékük `null` lesz (ha ezt próbáljuk használni, akkor `NullPointerException` kivételt kapunk), tehát referenciákról beszélünk. Például:

  ```
  int[] intArray = new int[5];
  ```

  A tömb elemei a létrehozás során a típusnak megfelelő alapértelmezett értéket kapják meg. Így például az objektumok a `null` referenciát, az `int` értékek a `0` értéket, a `boolean` értékek a `false` értéket, és így tovább.

  Inicializálásnál a `new` operátor hívása mellett használható az `{ e1, e2, ... }` forma, ahol `e1`, `e2` stb. az elemeket jelölik. Néhány példa tömbök inicializálására, ahol `barr1`, `barr2` és `barr3` egymással egyenértékű definíciók:

```
boolean[] barr1 = { true, false };
boolean[] barr2 = new boolean[] { true, false };
boolean[] barr3 = new boolean[2];
barr3[0] = true; barr3[1] = false;
```


Tömböket tömb értékű kifejezéssel is létrehozhatunk "ad hoc" módon:

```
public static int sum(int[] arr) { /* ... */ }

public static void main(String[] args) {
    int result = sum(new int[] { 1, 2, 3 });
    /* ... */
}
```

Clone repository (/javagyak/03/wikis/git_access)

Home (/javagyak/03/wikis/home)

- A tömb mindig tárolja a saját méretét, ez a `length` attribútumon keresztül lehet lekérdezni. Vegyük észre, hogy a mérete példányosítás után már nem változhat. Ha bővíteni akarjuk a tömböt, akkor egy új (nagyobb méretű) példányt kell létrehozni és oda átmásolni az elemeket.

```
int intArray[] = { 1, 2, 3, 4, 5 };

for (int i = 0; i < intArray.length; ++i) {
    System.out.println(intArray[i]);
}
```

Ezzel kapcsolatban érdemes megjegyezni, hogy tömbökre vonatkozó ciklusok szervezése esetén alkalmazható az ún. "foreach" alak, ahol sorban tudjuk járni az elemeket, ezért az indexváltozó elhagyható.

```
for (int x : intArray) {
    System.out.println(x);
}
```

- Ha hibás indexet adunk meg (vagyis a fenti képlet szerint a tömbhöz tartozó, összefüggő memóriaterületen kívülre mutató referenciát kapnánk), akkor ezt a futtató rendszer egy kivétellel jelzi. Ennek a neve `ArrayIndexOutOfBoundsException` .

Többdimenziós tömböket is tudunk készíteni. Ilyenkor csak a dimenziók számának megfelelően kell a deklarációban ismételnünk a tömbtípus képző `[]` jelölést. Például egy kétdimenziós tömböt így tudunk megadni:

```
int[][] arr;
```

Ilyenkor az inicializálásnál arra viszont ügyelni kell, hogy az első dimenziót meg kell adnunk:

```
int[][] arr = new int[5][];
```

A parancssori paraméterek mint tömb

Ahogy korábban láthattuk, a Java futtató rendszere a program belépési pontján a parancssori paramétereket is egy tömb formájában adja át. Ez a tömb `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) típusú értékeket tartalmaz, amelyek a

program neve után, szóközökkel elválasztott szavakat tartalmazza.

```
public static void main(String[] args) {
    System.out.println("Number of command line parameters: " + args.length);
    System.out.println("Their values are as follows:");

    for (int i = 0; i < args.length; ++i) {
        System.out.println(String.format("Parameter #%d: %s", i, args[i]));
    }
}
```


`String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékek mint tömbök

A Java nyelvben a karakterláncokat ábrázoló `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) objektumtípus közvetlenül nem viselkedik tömbként. A karakterláncok egyes karaktereit az osztály `charAt()` metódusának segítségével tudjuk elérni, azonban ennek meghívása nem minden esetben kényelmes. Ilyenkor lehet hasznos, ha a `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) objektumokat karakterek tömbjévé tudjuk alakítani a `toCharArray()` metódussal. Megjegyzendő, hogy a `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) osztálynak létezik olyan konstruálási módja, ahol karaktertömböt tudunk megadni, tehát a konverzió visszafele is alkalmazható.

Például cseréljük ki egy karakterláncban az összes `a` karaktert a `b` karakterre! Az eredeti szöveget azonban ne változtassuk meg, hanem hozzunk létre egy másikat, és azt adjuk vissza!

```
static String replaceAllAsToBs(String input) {
    char[] contents = input.toCharArray();
    char[] result = new char[contents.length];

    for (int i = 0; i < contents.length; ++i) {
        if (contents[i] == 'a') result[i] = 'b';
        else result[i] = contents[i];
    }

    return new String(result);
}
```

A tömbökhöz tartozó segédosztály

Érdemes tudni, hogy számos, tömbökkel kapcsolatos műveletet meg lehet találni a `java.util.Arrays` (http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html) osztályban. Ilyenek például a szöveggé alakítás (`toString()`), bináris keresés (`binarySearch()`), vagy a tömb feltöltése (`fill()`).

Például kérdezzük a parancssori paraméterként megadott számokat, majd írassuk ki ezeket a szabványos kimenetre az értékük szerint növekvő sorrendben!

```
public static void main(String[] args) {
    java.util.Arrays.sort(args);
    System.out.println(java.util.Arrays.toString(args));
}
```

Vegyünk azonban észre, hogy a fenti megoldás nem tökéletes. A parancssori paramétereket számként kell kezelni a rendezés során. Ezt a `sort()` függvény másik paraméterezésével lehet elérni.

```
public static void main(String[] args) {
    java.util.Arrays.sort(args, new Comparator<String>() {
        public int compare(String s1, String s2) {
            return (Integer.parseInt(s1) - Integer.parseInt(s2));
        }
    });
    System.out.println(java.util.Arrays.toString(args));
}
```


Itt egy ún. névtelen osztályt használtunk, erről majd a későbbiekben lesz szó.

Rendezzük a parancssori paramétereket a bennük szereplő `a` karakterek száma szerint!

```
public static void main(String[] args) {
    static int aCount(String s) {
        int result = 0;

        for (char c : s.toCharArray()) {
            if (c == 'a') result += 1;
        }

        return result;
    }

    java.util.Arrays.sort(args, new Comparator<String>() {
        public int compare(String s1, String s2) {
            return aCount(s2) - aCount(s1);
        }
    });
    System.out.println(java.util.Arrays.toString(args));
}
```

Tömbök összehasonlítása

Vigyázzunk, mivel a tömbök objektumtípust képviselnek, ezért közvetlenül az `==` (egyenlővizsgálat) operátor nem azt az eredményt adja, amit várnánk! Mivel a tömbök azonosítói valójában objektumokra hivatkozó referenciák, ezért a tartalmuk összehasonlítására vagy a `java.util.Arrays` (http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html) osztály valamelyik megfelelő metódusát alkalmazzuk (`equals()` , esetleg `deepEquals()` a többdimenziós esetben), esetleg implementáljuk saját magunk.

Például egy függvény segítségével ellenőrizzük, hogy egy kilencelemű tömb tartalmazza-e az összes számot 1 és 9 között!

```
static boolean isValid(int[] input) {
    int[] reference = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    java.util.Arrays.sort(input);
    return java.util.Arrays.equals(input, reference);
}
```

Stringek

A `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékek a tömbökhöz hasonlóan ugyanúgy objektumtípusok, azonban a szintaxisuk nem annyira speciális. Leginkább a hozzájuk tartozó literálok, valamint az összefűzés műveletei lettek leegyszerűsítve.


```
String s = "Hello" + " " + "there" + "!";
```


Természetesen, mivel objektumtípus, ezért a fentebb létrehozott `s` változó egy referencia lesz. Ha nem inicializáltuk volna, akkor az értéke `null` lett volna, amelynek a feloldása a `NullPointerException` kivételhez vezet. Ugyanígy, az összehasonlításnál ügyelni kell arra, hogy az `==` operátor mindig csak a referenciákat hasonlítja össze. Helyette használjuk az `equals()` metódust!

A `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) osztályban sok hasznos metódus lehet találni, amelyekkel könnyebbé válik a szöveg feldolgozása. Ezek közül korábban már láthattuk a `charAt()` és `toCharArray()` metódusokat. A teljesség igénye nélkül bemutatunk még néhány további hasznos metódust.

- Hossz lekérdezése, üresség vizsgálata.

```
int length = s.length();
boolean empty = s.isEmpty();
```

- Annak eldöntése, hogy az érték adott szövegrészlettel kezdődik vagy fejeződik be.

```
boolean isExecutable = s.endsWith(".exe");
boolean isPre = s.startsWith("pre_");
```

- Karakter előfordulása valahol a szövegben. Az `indexOf()` metódus az adott karakter első előfordulását keresi meg és amennyiben az megtalálható, visszaadja a neki megfelelő indexet.

```
int aIndex = s.indexOf('a');
boolean hasExclamantion = s.contains('!')
```

- Szövegrészlet keresése.

```
String snippet = s.substring(1, 3);
```

- Szöveg nagybetűsítése. Itt látható, hogy a `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékek tulajdonképpen sosem változnak meg, hanem mindig új változat jön létre belőlük.

```
String upper = s.toUpperCase();
```

- Szöveg felosztása valamilyen határolókarakterek mentén.

```
String[] words = s.split(" ");
```

- Szövegrészlet kicserélése. Ennek alapja az ún. reguláris kifejezések alkalmazása. Ez egy olyan miniatűr nyelv, amelyben illeszkedési mintákat tudunk szerkeszteni a `+` és `*` operátorok segítségével. Ezeket mindig egy karakterhez kapcsoljuk és így tudjuk leírni, hogy egy vagy több olyan karakterre számítunk (`+`), vagy nem kötelező szerepelnie az adott pozíción ahhoz (`*`). Minden más esetben szöveg szerinti egyezést vizsgálunk.

  Tehát az `a+` minta `a` karakterek végtelen sorozatát jelenti (vagyis ameddig lehetséges rá a bemenet), ahol legalább egy `a` karakternek lennie kell. Ezt használjuk fel ebben a példában is.

Clone repository (/javagyak/03/wikis/git_access)

Home (/javagyak/03/wikis/home)

```
public class Replace {
  public static void main(String[] args) {
    for (String s : args) {
//    convert(s);
//    s = s.replace("a+", "a");
      s = s.replaceAll("a+", "a");
      System.out.println(s);
    }
  }

  // aaaaabaaaaaacaaaaada -> abacada
  static String convert(String s) {
    String result = "";
    // Megkeressük az első 'a' karaktert és addig folyamatosan
    // tároljuk a többit.
    ...
    // Elmegyünk az 'a' karakterek végéig
    ...
    // Ismételjük az eddigieket, amíg a bemeneti paraméter végére nem
    // érünk.
  }
}
```

`String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékek kezelése hatékonyabban

A szövegrészletek cserélése kapcsán láthatjuk, hogy a `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) tulajdonképpen egy ún. "perzisztens" implementációval rendelkezik. Ennek az a lényege, hogy már a kontenáció során sem változik meg egyik komponens értéke sem, hanem egy új, az előzőektől független `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) érték keletkezik. Nézzük meg az alábbi példában:

```
String string = "AAAxAAA";
string.replace('x', 'A');
System.out.println(string);         // --> "AAAxAAA"
string = string.replace('x', 'A');
System.out.println(string);         // --> "AAAAAAA"
```

Ez a jellemző az erőforrások kihasználása tekintetében nem teszi eléggé hatékonnyá ezt a típust. Így olyankor, amikor egy `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értéket szeretnénk gyors egymásutánban megváltoztatni, javasolt a `StringBuffer`

(http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html) használata. Ennek az osztálynak számos olyan metódusa van, amely a szövegekkel kapcsolatos gyakori műveleteket hatékonyan valósítja meg.

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
sb.reverse();
System.out.println(sb.toString());  // --> "dlroW olleH"
sb.reverse();
sb.setCharAt(6, '-');
System.out.println(sb.toString());  // --> "Hello-World"
sb.deleteCharAt(6);
System.out.println(sb.toString());  // --> "HelloWorld"
sb.delete(0, sb.length());
System.out.println( sb.toString()); // --> ""
```


Látható, hogy a `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) és `StringBuffer` (http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html) típusok között tudunk oda-vissza konvertálni. Az `append()` metódus a puffer aktuális tartalmához hozzáfűzik a paraméter értékét, a `reverse()` megfordítja a puffer tartalmát, a `setChar()` beállítja a puffer adott indexű karakterét a második paraméterként megadottra. A `deleteChar()` törli az adott indexű elemet, a `delete()` pedig az adott indexek intervallumában törli egy részét a puffernek. Itt is alkalmazható az aktuális hossz lekérdezésére a `length()` metódus.

A `java.util.Scanner` (http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html) osztály

A `java.util.Scanner` (http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html) osztály segítségével `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) típusú értékek feldolgozását könnyíthetjük meg. Leginkább akkor tudjuk hasznosítani, amikor egy `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékből kell kiolvasni más típusú értékeket, vagy esetleg szóközökkel tagolt szavakat akarunk egyetlen `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékből kinyerni. Alapvetően a szabványos bemenet is felfogható egy végtelen hosszúságú `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értéknek, ezért annak programozásakor remekül tudjuk ezt az osztályt alkalmazni.

Például olvassunk be a szabványos bemenetről két szót! Itt a `next()` metódus mindig a következő szót (szóközhatárig) olvassa be. A `hasNext()` metódus segítségével lehet megállapítani, hogy a feldolgozandó forrásban áll-e rendelkezésre következő elem. Ezt mindig tanácsos meghívni a `next()` metódus előtt.

```
public static void main(String[] args) {
    java.util.Scanner sc = new java.util.Scanner(System.in);

    while (sc.hasNext()) {
        String s = sc.next();
        System.out.println("There was a word: " + s);
    }
}
```

Hasznos tudni, hogy a `java.util.Scanner` (http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html) akár `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékek esetén is működik. Például olvassunk ki egy `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékben rejtő számokat, amelyeket szóközök választanak el! Megfigyelhetjük, hogy a számok kiolvasását is megelőzi egy vizsgálat.


```
public static void main(String[] args) {
    java.util.Scanner sc = new java.util.Scanner("42 99 -1 0 32");

    while (sc.hasNextInt()) {
        int x = sc.nextInt();
        System.out.println("There was a number: " + x);
    }
}
```

Clone repository (/javagyak/03/wikis/git_access)

Home (/javagyak/03/wikis/home)

Feladatok

- Készítsünk egy `utils.NumericArrays` osztályt, amelyben hozzuk létre a következő statikus metódusokat mint eljárásokat (vagyis a paraméterként kapott tömböket mindig megváltoztatjuk):
  - egy `sum()` függvényt, amely kiszámítja egy számokat tartalmazó tömb elemeinek összegét!
  - egy `average()` függvényt, amely kiszámítja egy tömb elemeinek az átlagát!
  - egy `normalize()` függvényt, amely normálja egy számokat tartalmazó tömb elemeit! A normálás azt jelenti, hogy az elemek összegének egynek kell lennie.

  Adjunk az osztályhoz egy főprogramot, amely egy tömb értékeit és a velül elvégzendő művelet (`"sum"` , `"avg"` , `"norm"`) parancssori paraméterként kapja meg!
- Készítsünk egy `utils.Cryptor` osztályt, amely tartalmazza a következő statikus metódusokat:
  - egy `encode()` nevű kódoló függvényt, amely egy `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékként megadott szöveget (mint paraméterét) úgy kódolja, hogy minden karakterrel és egy konstans értékkel XOR (kizáró vagy) műveletet végez! (Ehhez használjuk az `^` operátort!)
  - egy `decode()` nevű függvényt, amely az előző függvény inverze, amellyel a konstans érték ismeretében vissza tudjuk kódoló az előző függvénnyel kódolt értékeket!
- Egészítsük ki az előbbi, `utils.Cryptor` osztály metódusait úgy, hogy a kódolást és a dekódolást ne egyetlen konstanssal, hanem egy másik `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékkel legyen képesek elvégezni!
- Készítsünk egy `utils.Sudoku` osztályt, amelyben szerepeljenek a következő statikus metódusok:
  - egy olyan `check()` nevű függvény, amely ellenőrzi, hogy egész értékeket tartalmazó, 3x3 elemű kétdimenziós tömb értékei csak 1 és 9 között szerepelnek, illetve mindegyik számot csak egyszer tartalmazza!
  - egy olyan `show()` nevű függvényt, amely előállítja egy kétdimenziós tömbnek egy olyan szöveges reprezentációját, amely mátrixként formázva tartalmazza annak elemeit!
- Készítsünk egy `utils.Vector` osztályt, amelyben szerepeljenek a következő statikus metódusok:
  - egy olyan `scalarProduct()` nevű függvényt, amely képes meghatározni két paraméterként kapott, valós számokat tartalmazó tömb (mint vektorok) skaláris szorzatát!
  - egy olyan `vectorialProduct()` nevű függvényt, amely képes meghatározni két paraméterként kapott (valósokat tartalmazó) tömb vektoriális szorzatát az általuk bezárt szög alapján! Az eredményt

három dimenzióban számítsuk ki.

- Készítsünk egy `utils.Matrix` osztályt, amelyben szerepeljenek a következő statikus metódusok:
  - egy olyan `scalarProduct()` nevű függvényt, amely képes meghatározni egy paraméterként kapott, kétdimenziós tömbként ábrázolt mátrix és egy valós szám szorzatát!
  - egy olyan `add()` nevű függvényt, amely képes meghatározni két paraméterként kapott, kétdimenziós tömb mátrix összegét!
  - egy olyan `multiply()` nevű függvényt, amely képes meghatározni két paraméterként kapott, kétdimenziós tömb mátrix szorzatát!
- Készítsünk egy olyan programot, amely egy parancssori argumentumként megadott `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékét átalakít a következő módokon:
  - minden numerikus karaktert változatlanul hagy,
  - minden betűt kisbetűvé alakít,
  - minden egyéb karaktert lecserél egy `_` (aláhúzásjel) karakterre,

  majd az eredményt kiírja a szabványos kimenetre.
- Készítsünk egy programot, amely minden ékezetes karaktert lecserél a neki megfelelő, ékezet nélküli változatára! A bementetet ezúttal is parancssori argumentumként adjuk meg és az eredményt pedig a szabványos kimenetre írjuk ki.
- Készítsünk egy olyan függvényt, amely a paramétereként megadott `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) értékben található kisbetűvel kezdődő szavakat nagybetűssé alakítja! A visszatérési értéke legyen a megváltoztatott `String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html)!

Linkek