Java tutorial

Tömbök, kollekciók és egyéb alaposztályok

Néhány alaposztály, amit ismerni illik

- Object
- String
- StringBuffer
- Integer, Double, ...
- System
- Math

Object

- Minden más osztálynak az őse. (Alapértelmezett, ha nincs **`extends`**.)
- Tehát minden objektum beletartozik az Object osztályba.
- Alapvető műveleteket definiál, amelyekkel minden objektum kell, hogy rendelkezzen.

`String toString()` stringgé alakít

`boolean equals( Object o )` összehasonlít

egyebek: clone, getClass, hashCode, finalize, stb.

toString

- Tetszőleges objektumra működik:

```
System.out.println( "o = " + o );
```

- A + jel hatására az `o` automatikusan `String`-gé konvertálódik, mégpedig a `toString` meghívásával.
- Sok osztály definiálja a `toString` specializálásával, hogy hogyan kell szöveges formára alakítani az objektumait.
- Az `Object`-beli alapértelmezett működés az objektum dinamikus típusának nevét és `hashCode`-ját írja ki. (Ez utóbbi egy szám.)

```
System.out.println(new String[2]);
[Ljava.lang.String;@23bcc1
```

equals vs. ==

- Objektumok egyenlőségét az `equals` metódussal illik vizsgálni. Alapértelmezésben ez ugyanaz, mint az `==`.
- De például mikor egyenlő két dátum? Ha ugyanazt az időpontot jelölik, nem?
- Az `==` azt mondja meg, hogy két referencia mikor egyenlő. (Azaz, hogy a két objektum mikor **azonos**.) Például **nem lesz igaz** az alábbi:

```
new Dátum(1970,4,28) == new Dátum(1970,4,28)
```

- Az a szokás, hogy az osztályok, felüldefiniálva az `equals` metódust, megadják, hogy mikor **egyenlő** két objektum. Ez már **igaz** lehet:

```
(new Dátum(1970,4,28)).equals( new Dátum(1970,4,28) )
```

Az equals metódus a Dátum osztályban

```
public class Dátum {
  int év, hónap, nap;
  public boolean equals( Object d ){
    if( d instanceof Dátum )
       return év    == ((Dátum)d).év &&
              hónap == ((Dátum)d).hónap &&
              nap   == ((Dátum)d).nap;
    else return false;
  }
  public int hashCode(){   // ez is kell
    return 400*év + 31*hónap + nap;
  }
  ...
}
```

Példa dátumokkal

```
Dátum d1 = new Dátum(2001,1,1);
Dátum d2 = new Dátum(2001,1,1);
Dátum d3 = new Dátum(1789,7,14);
d1.equals(d2) ?
d1.equals(d3) ?
d1 == d2 ?
d1 == d3 ?

(new Object()).equals( new Object() ) ?
     new Object() == new Object() ?
```

Java tutorial

A String osztály

- Unicode karakterek sorozata

```
String s = "Szia!";
String s = new String("Szia!");
```

- Egy String objektum tartalmát nem lehet módosítani. Helyette új stringet kell létrehozni.

```
s = "Szia?";
s = new String("Szia?");
```

- De rengeteg konstruáló, lekérdező és összehasonlító művelet van.

`length, charAt, compareTo, concat, endsWith, replace, substring, trim, valueOf, indexOf, equalsIgnoreCase, toLowerCase, ...`

A StringBuffer osztály

- Unicode karakterek sorozata
- A tartalmát megváltoztathatom anélkül, hogy új objektumot kellene létrehozni.
- Más a reprezentációja, mint a String osztálynak, ezért más műveleteket lehet rajta hatékonyan megvalósítani.

`append, insert, reverse, setCharAt, setLength`

- A műveletek, azon kívül, hogy transzformálják az objektumot, vissza is adnak egy referenciát rá. Ez ezért jó:

```
StringBuffer b = new StringBuffer("Szia!");
b.append(" Hi!").append(" Salut!").reverse();
```

Csomagoló osztályok

- A primitív típusokhoz léteznek csomagoló (wrapper) osztályok: `Integer`, `Byte`, `Double`, `Boolean`, stb.

```
int i = 42;
Integer j = new Integer(i);
```

- Akkor használjuk őket, ha primitív típusú értéket akarunk objektumként használni. Például, ha a `toString` kéne...

```
System.out.println("minden=" + 42);
```

Ehhez ez kell:

```
new Integer(42).toString()
```

Egy másik fontos példa kicsit később...

```
Vector v = new Vector();      v.add(j);
```

Csomagoló osztályok (2)

`boolean`	`Boolean`	`byte`	`Byte`
`char`	`Character`	`short`	`Short`
`int`	`Integer`	`long`	`Long`
`float`	`Float`	`double`	`Double`

- Műveletek pl. az `Integer` osztályhoz:

```
static int parseInt(String s [, int radix])
Integer( String s )
static String toString( int i [, int radix])
final static int MIN_VALUE, MAX_VALUE
```

Még néhány érdekes osztály

- Math
  - `Math.PI, Math.E`
  - `abs(), sqrt(), pow(), log(), min(), floor(), round(), rint(), random(), sin(), asin(), toRadians()`
- System
  - `in, out, err`
  - `currentTimeMilis(), exit(int), gc()`
- Number
  - A szám jellegű csomagoló osztályok közös őse.

Feladat

- A Java nyelv specifikációja megengedi a fordítóknak, hogy ne hozzanak létre azonos string-literálokhoz külön objektumokat. Próbáld ki, hogy az általad használt fordító milyen!
- És mi a helyzet két `Integer` objektummal?

A String-es feladathoz:

```
String s = new String("Szia!");
String z = new String("Szia!");
if( s == z ) ...
```

Java tutorial

Tömbök

- Sok, azonos típusba tartozó érték tárolására
- Hatékony elérés: indexelés
- Időigényes beszúrás és törlés

Egy régi példaprogram fel-e-le-ve-ní-té-se

```
class Hossz {
  public static void main( String args[] ){
    if( args.length > 0 )
      System.out.println(args[0].length());
  }
}
```

Tömb típusú változó definíciója

```
int t[];
int[] t;
```

Tömb létrehozása

- A változó deklarációja nem hozza létre a tömböt.
- A tömbtípusok olyanok, mint az osztályok.
  A tömbök lényegében objektumok.

```
int[] t = new int[10];
int t[] = new int[10];
```

- Egy tömb típusú változó csak egy referencia.

```
int[] s;
int x = s[0]; int y = s.length;
```
Hibásak!

Amíg nincs létrehozva a tömb objektum, addig nem megy!
`NullPointerException`

```
int[3] x; int y[3];
```
Hibásak!

A tömb elemeinek elérése

- A tömbök nullától indexelődnek, hossz mínusz egyig.
- A hosszt a tömb létrehozásakor adjuk meg.
  - A hosszt megváltoztatni nem lehet.
  - A referenciának viszont mindegy, hogy milyen hosszú a tömb.

```
int[] x = new int[5];
x = new int[7];
int y[] = x;
```

- A hossz lekérdezése: `args.length`
- A tömb indexelése: `args[i]`
- A futtató rendszer ellenőrzi az indexhatárokat!
  Ha nem stimmel: `ArrayIndexOutOfBoundsException`

Tömb feltöltése

```
double[] t = new double[3];
t[0] = 1.0;
t[1] = 2.0;
t[2] = 3.0;
```

```
double[] s = new double[3];
for( int i = 0; i < s.length; i++ )
  s[i] = i;
```

```
double[] v = {1.0, 2.0, 3.0};
```

```
double[] w;
w = new double[] {1.0, 2.0, 3.0};
```

Objektumok tömbje

Igazándiból referenciákból álló tömb.

```
Pont[] t;
t = new Pont[3];
for (int i=0; i<3; i++)
  t[i] = new Pont(i,2);
```

```
Pont[] s = { new Pont(1,2),
             new Pont(2,2),
             new Pont(3,2) }
```

Java tutorial

Feladat

- A Verem osztály mintájára készítsd el a Sor osztályt! A sor Object-eket tárol, műveletei a put és a get. A reprezentációhoz használj tömböt, melynek méretét a konstruktornak kell átadni paraméterként.
- Segítség: tarts nyilván két indexet, amelyek azt jelzik, hogy a soron következő get, ill. put művelet melyik elemen kell dolgozzon. A két index járjon körbe-körbe a tömbön. Kivételkezeléssel (üres vagy tele a sor) ne foglalkozz!

Verem osztály

```
public class Verem {

  Object[] adatok;
  int veremtető = 0;
  int maxméret;

  public Verem( int maxméret ){
    this.maxméret = maxméret;
    adatok = new Object[maxméret];
  }

  public void push( Object o )
      adatok[veremtető] = o;
      veremtető ++;
  }

  public Object pop(){ ... }

}
```

Többdimenziós tömb

- Nincs Java-ban.
- De lehet csinálni **tömbök tömbjét**.

```
int[][] mdt = new int[3][2];
for (int i=0; i<3; i++)
  for (int j=0; j<2; j++)
     mdt[i][j] = i*j;
```

Szabálytalan alakú "mátrix"

```
int mdt[][];
mdt = new int[2][];

mdt[0] = new int[3];
mdt[0][0] = 7;
mdt[0][1] = 2;
mdt[0][2] = 9;

mdt[1] = new int[4];
mdt[1][0] = 2;
mdt[1][1] = 4;
mdt[1][2] = 8;
mdt[1][3] = 0;
```

Java tutorial

Heterogén tömb

- Egy tömbben többféle osztályú objektumot tárolhatok
- Ha: rendelkeznek valamilyen közös őstípussal
- A tömbelemek statikus típusa közös. Ez mondja meg, hogy milyen műveleteket használhatunk rájuk.
- A tömbelemek dinamikus típusa különböző lehet. Ez mondja meg, hogy melyik implementációt kell végrehajtani.

```
Alkalmazott t[] = new Alkalmazott[100];
for( int i=0; i<100; i++ )
   t[i] = new Alkalmazott();
t[42] = new Fonok();

for( int i=0; i<100; i++ )
   System.out.println( t[i].fizetés() );
```

Tömbtípus, mint osztály

- Sok szempontból olyan...
- "Altípusosság"

```
Object[] t = new Alkalmazott[3];
```

- Vigyázat! Ezután futási idejű hibát okoz:

```
t[0] = new Object();
```

- Ez viszont jó lesz:

```
t[0] = new Fonok();
```

Feladat

Valósítsd meg a Mátrix típust.

Elemek típusa: double

Műveletek:

- Létrehozás (nullákkal, illetve egy adott elemmel feltöltve)
- Elem lekérdezése és beállítása
- Skalárral való szorzás
- Hozzáadás, összeadás
- Szorzás (Házi feladat)
- egységmátrix létrehozása (négyzetes)
- toString

A reprezentációhoz használj kétdimenziós tömböt.

Java tutorial

Egyéb kollekciók

- sorozat
- halmaz
- verem
- kupac
- gráf
- vektor
- lista
- sor
- fa
- hash-tábla

Hatékonyság

- "Trade-off"
- Tömb: konstans elérés, lineáris beszúrás
- Lista: lineáris elérés, konstans beszúrás
- Kupac: logaritmikus elérés és beszúrás
- Válasszuk azt, ami a feladathoz a leginkább megfelel.

Lehetőségek Java-ban

- Tömbök
- A `Vector`, `Stack` és `Hashtable` osztályok a `java.util` csomagból
  - A JDK 1.0 óta vannak már a nyelvben
- Gyűjtemény Keretrendszer (Collections Framework)
  - A JDK 1.2 óta szabványosított része a nyelvnek
- Egyéb, nem standard könyvtárak is elérhetők, pl. JGL.

Vector

- Viszonylag gyors lekérdezés/beszúrás/törlés eleje, vége, közepe
- Object-eket tárolhatunk benne

```
boolean contains( Object elem )
int indexOf( Object elem )
Object elementAt( int index )
Object lastElement()
Object removeElementAt(int index)
boolean removeElement( Object elem )
void insertElementAt( Object elem, int index )
void addElement( Object elem )
```

stb.

Példa a Vector használatára

```
import java.util.*;
class RövidSzavak {
  public static void main(String args[]){
    Vector v = new Vector();
    for( int i=0; i<args.length; i++ ){
      if( args[i].length() < 10 ){
        v.add(args[i]);
      }
    }
  }
}
```

A Vector elemei

- Az elemek Object típusúak.

```
void utolsóBorNélkül( Vector v ){
  for( int i=v.size()-1; i>=0; i-- ){
    String s = (String) v.getElementAt(i);
    if( s.endsWith("bor") ){
       v.removeElementAt(i);
       return;
    }
  }
}
```

Java tutorial

Számok sorozata

- Ha tömböt szeretnénk használni:

```
int[] t = {2,42,37};                new int[3]
Integer[] t = { new Integer(2),
       new Integer(42), new Integer(37) };
```

  – Egész másképp helyezkednek el a memóriában az adatok.

- Ha mást, mondjuk `Vector`-t:

```
Vector v = new Vector();
v.add( new Integer(42) );
v.add( new Integer(37) );
v.insertElementAt( new Integer(2), 0 );
```

Enumeration

- Iterátor, melynek segítségével egy gyűjtemény bejárható. Például egy Vector...

```
void kiír( Enumeration e ){
  while( e.hasMoreElements() ){
    Object o = e.nextElement();
    System.out.println(o);
  }
}

kiír(v.elements())
```

Java tutorial

Kapacitás

- Akármennyi elemet tárolhatunk a Vector objektumokban.
- A pozícionálós műveletek hatékonyságának biztosítása érdekében "tömb jellegű" belső reprezentáció.
- Időnként "megtelik": ilyenkor szerez még egy kis memóriát, hogy tovább lehessen belepakolni.
- Automatikusan megnöveli a **kapacitását**.
- A kezdeti kapacitás beállítható a konstruktorban.

```
Vector()      Vector( int initialCapacity )
Vector( int initialCapacity,
        int capacityIncrement )

capacity(), ensureCapacity(), trimToSize()
```

Feladat

- Írj egy műveletet, amely egy vektorból kiszedi azon elemek első előfordulásait, amely elemek többször is előfordulnak a vektorban. Az alábbi módon valósítsd ezt meg. Lekérve egy Enumeration objektumot, pakold át sorra az elemeket egy másik vektorba, ha még nem voltak benne. Ha már benne voltak, akkor egy harmadik vektorba tedd őket, ha abban még nem voltak. Ezután ezt a harmadik vektort járd be egy Enumeration segítségével, és minden elemének töröld ki az első előfordulását a legelső vektorból.

Stack: verem adattípus

- A Vector leszármazottjaként lett definiálva.
- Szokásos veremműveletek:

```
Object push( Object elem )
Object pop()
Object peek()
boolean empty()
```

Hashtable

- Ha nem egész számokkal akarunk indexelni, hanem tetszőleges objektumokkal...
- kulcs - érték párokat tárolhatunk

```
Hashtable ht = new Hashtable();
ht.put("kalap", "piros");
ht.put("kabát", "kék");
String kalapSzíne = (String) ht.get("kalap");
```

- Egy pár beszúrása és visszakeresés kulcs alapján:

```
Object put( Object key, Object value)
Object get( Object key )
```

  - A kulcs is és az érték is akármilyen objektum lehet.
  - Egy táblán belül is lehet többféle kulcs...

További Hashtable műveletek

- keresések, törlés, kulcsok és értékek lekérdezése, stb.

```
boolean containsKey ( Object key )
boolean containsValue ( Object value )
Object remove ( Object key )
Enumeration keys()
Enumeration elements()
int size()
void clear()
```

- Hatékony tárolás és keresés, a kulcsok `hashCode()`-ja alapján.

Példa a Hashtable alkalmazására

- Megszámoljuk, hogy a parancssori argumetumok közül melyik szó hányszor szerepel.

```
java Hányszor elmegyek elmegyek messze
  megyek

{messze=1, elmegyek=2, megyek=1}
```

```
import java.util.*;
class Hányszor {
  public static void main(String[] args){
    Hashtable h = new Hashtable();
    for( int i=0; i<args.length; i++ ){
      String kulcs = args[i];
      Integer hányszor;
      if( h.containsKey(kulcs) ){
          hányszor = (Integer) h.get(kulcs);
          int megEgy = hányszor.intValue() + 1;
          hányszor = new Integer(megEgy);
      } else
          hányszor = new Integer(1);
      h.put( kulcs, hányszor );
    }
    System.out.println(h);
  }
}
```

Feladat

- A parancssori argumentumokban számokat adunk át a programunknak. Minden előforduló számhoz tároljuk el, hogy hányadik parancssori argumentum. Ha többször is előfordul a szám, akkor az összes előfordulásának sorszámát el kell tárolni. Ez alapján a nyilvántartás alapján a programunk írja ki a képernyőre, hogy melyik szám hányadik pozíciókon fordult elő.

```
java Kigyűjt 11 32 1 32 5 11 32

32: 1 3 6
5: 4
1: 2
11: 0 5
{32=[1, 3, 6], 5=[4], 1=[2], 11=[0, 5]}
```

Java tutorial


```
import java.util.*;
class Kigyűjt {

  public static void main(String[] args){
    Hashtable h = new Hashtable();
    for( int i=0; i<args.length; i++ )
      betesz( h,
              new Integer(i),
              new Integer(args[i]) );
    kiír(h);
    // System.out.println(h);
  }

  public static void betesz( Hashtable h,
                             Integer poz,
                             Integer szám ){...}

  public static void kiír( Hashtable h ){...}
}
```

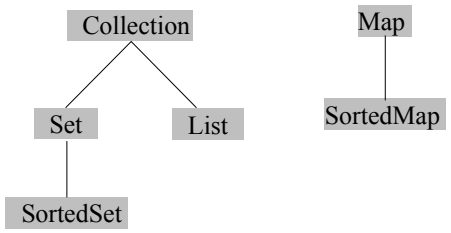
```
public static void betesz( Hashtable h,
                           Integer poz,
                           Integer szám ){

    Vector v;
    if( h.containsKey(szám) )
       v = (Vector) h.get(szám);
    else
       v = new Vector();
    v.add(poz);
    h.put( szám, v );

}
```

```
public static void kiir( Hashtable h ){
    Enumeration számok = h.keys();
    while( számok.hasMoreElements() ){
      Integer szám = (Integer)
                     számok.nextElement();
      System.out.print(szám + ": ");
      Vector v = (Vector) h.get(szám);
      Enumeration pozíciók = v.elements();
      while( pozíciók.hasMoreElements() )
        System.out.print(
            pozíciók.nextElement() + " "
        );
      System.out.println();
    }
}
```

A Gyűjtemény Keretrendszer

Gyűjtemények rendszere, absztrakciós szintek bevezetésével.

Interfészek: gyűjtemény, leképezés, halmaz, lista



Collection: gyűjtemény absztrakció

```
boolean add(Object o)
boolean contains(Object o)
boolean remove(Object o)

boolean addAll(Collection c)
boolean containsAll(Collection c)
boolean removeAll(Collection c)
boolean retainAll(Collection c)

void clear()
boolean isEmpty()
int size()              Iterator iterator()
Object[] toArray()
```

List: sorozat adattípus

- ugyanaz az elem többször is szerepelhet
- az elemek sorrendje lényeges
- két lista egyenlő, ha ua. elemek ua. sorrendben

- olyasminek képzelhetjük, mint a Vector
  sőt: a Vector megvalósítja ezt az interfészt
- rövidek a műveletek nevei: `set, get, add`

- a `remove` az elem legelső előfordulását távolítja el
  az `add` és `addAll` a sorozat végéhez adja hozzá

- kétféle megvalósítás: `LinkedList` és `ArrayList`

A List megvalósításai

- kétféle megvalósítás
  - láncolt (lista) a `LinkedList` osztály
    - hatékony a beszúrás a sorozat elejére
    - hatékony törlés bejárás közben
  - vektor jellegű az `ArrayList` osztály
    - hatékony pozícionáló műveletek
    - általában ezt használjuk (általában ez hatékonyabb)
    - a leszármazottja a `Vector`

Java tutorial

Egy példa a List interfészhez

```
import java. util.*;
public class ListábaPakolok {
  public static void main( String[] args) {
    List s = new ArrayList();
    s.add("Route");
    s.add(new Integer(66));
    s.add("Route");
    s.add("to LA");
    System.out.println(s);
  }
}
```

- `[Route, 66, Route, to LA]`

Set: halmaz adattípus

- egy elem csak egyszer szerepelhet a halmazban
- az elemek sorrendje indifferens
- két halmaz egyenlő, ha ua. az elemek vannak bennük
- tartalmazás: `contains, containsAll`
  únió: `addAll`
  metszet: `retainAll`
  különbség: `removeAll`
- Megvalósítás: `HashSet`
- Egy másik megvalósítás: `TreeSet`, ami a `SortedSet`-et is megvalósítja
  - Azaz a halmaz elemeihez rendezetten is hozzáférünk.

Egy példa a Set interfészhez

```
import java. util.*;
public class HalmazbaPakolok {
  public static void main( String[] args) {
    Set s = new HashSet();
    s.add("Route");
    s.add(new Integer(66));
    s.add("Route");
    s.add("to LA");
    System.out.println(s);
  }
}
```

```
[to LA, Route, 66]
```

Java tutorial

Iterator: gyűjtemény bejárása

- Olyasmi, mint az `Enumeration` interfész
  - picit rövidebbek a műveletek nevei
  - lehet törölni is bejárás során
- Műveletek:

`boolean hasNext(), Object next(), void remove()`

- Leszármazottja a `ListIterator`. Ilyen a sorozatoknak (`List`) van. Még több művelet...

```
boolean hasPrevious(), Object previous()
void add(Object), void set(Object)
int nextIndex(), int previousIndex()
```

Példa az Iterator használatára

```
import java.util.*;
public class Iteráció {
  public static void main(String[] args){
    Collection c = new ArrayList();
    for( int i=0; i<10; i++)
      c.add( new Double(Math.random()) );
    Iterator it = c.iterator();
    while( it.hasNext() ){
      Double d = (Double) it.next();
      if( d.doubleValue() < 0.6 )
        it.remove();
    }
    System.out.println(c);
  }
}
```

Map: leképezés adattípus

- Olyasmi, mint a HashTable
- kulcs és érték párok
- Iteráció
  - kulcsokon: `keySet().iterator()`
  - értékeken: `values().iterator()`
  - párokon (`Entry` a neve): `entrySet().iterator()`
- Két implementáció: `HashMap` és `TreeMap`
  - A `TreeMap` megvalósítja `SortedMap`-et is (rendezettség!)
- Általában a `HashMap`-et használjuk, az hatékonyabb.

Java tutorial

Rendezés

- Tömböket: az `Arrays` osztály `sort` műveletével
- Sorozatokat, azaz `List`-eket a `Collections` osztály `sort` műveletével.
- Ha a tömbben/sorozatban tárolt elemeken van természetes rendezés. (Lásd a `Comparable` interfészt!)
  - `int[], Integer[]`
  - `List`, amibe `Double` objektumokat teszünk
- Lehet saját rendezést definiálni, és aszerint rendezni. (Lásd a `Comparator` interfészt!)

Természetes rendezés szerint

- `Comparable: int compareTo(Object o)`

```
import java.util.*;
class Rendez {
  public static void main( String[] args )
  throws ClassCastException {
    Arrays.sort(args);
    for( int i=0; i<args.length; i++ )
      System.out.println(args[i]);
  }
}
```

```
  Hello
java Rendez Szia Hello Salut                Salut
```

Feladat

- Tegyük fel, hogy a parancssori argumentumok egész számok. Írjuk ki rendezve őket!

Saját rendezés definiálása

- **`Comparator: int compare( Object o1, Object o2 )`**

```
class HosszHasonlító implements Comparator {
  public int compare(Object o1, Object o2) {
    int h1 = ((String) o1).length();
    int h2 = ((String) o2).length();
    if( h1<h2 )   return -1;
    if( h2<h1 )   return 1;
    return 0;
  }
}
```

```
import java.util.*;
class Rendez {
  public static void main( String[] args )
  throws ClassCastException {
    Arrays.sort(args, new HosszHasonlító());
    for( int i=0; i<args.length; i++ )
      System.out.println(args[i]);
  }
}
```

```
java Rendez Szia Hi Salut
```

```
Hi
Szia
Salut
```

Rendezés sorozatokon

- Ugyanígy, csak a `Collections.sort` műveletet kell meghívni egy `List` objektumon.
- Akár természetes rendezés, akár saját rendezés lehet.

Egyéb műveletek

- minimum- és maximumkeresés min(), max()
- bináris keresés binarySearch()
- megfordítás reverse()
- feltöltés, másolás fill(), copy()
- rendezés megszüntetése shuffle()

- Arrays, Collections

Feladat

- Oldd meg a számok rendezését másképp: írj egy olyan `Comparator` osztályt, ami `String`-eket úgy hasonlít össze, hogy a `parseInt` művelettel először számot csinál belőlük, és azután ezeket hasonlítja össze.
- Oldd meg a számok rendezését úgy, hogy a parancssori argumentumokat nem tömbbe pakolod, hanem egy `ArrayList` objektumba!

Java tutorial