Programozási Ismeretek

Dr. Gergely Tamás
Dr. Ferenc Rudolf, Dr. Jász Judit, Dr. Kiss Ákos

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék

2024

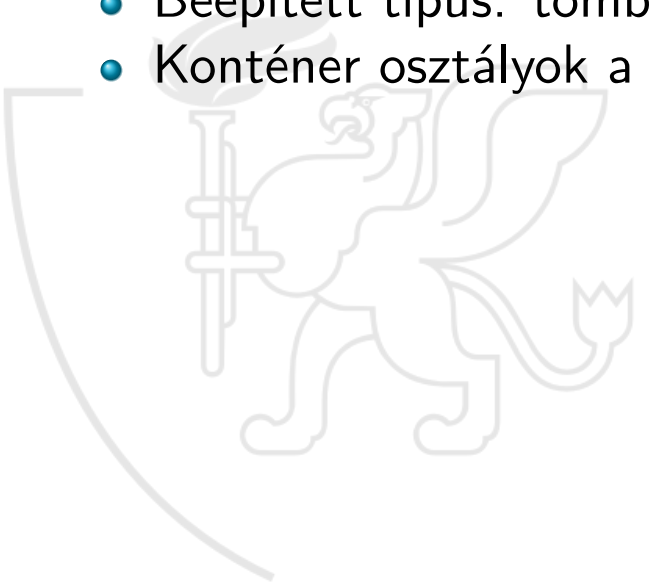
(v0214)

Tartalom

Tartalom

Objektumok tárolása

- Csak nagyon egyszerű program esetében van előre meghatározott számú objektum ismert élettartammal
- Általában ez csak futási időben derül ki
  - kell, hogy legyen lehetőség tetszőleges számú objektumot létrehozni bármikor, bárhol
  - ezeknek nem lehet mindig egyedi nevet adni...
- A megoldás: *Kollekciók* (*konténerek*)
  - Beépített típus: tömb (array)
  - Konténer osztályok a `java.util` könyvtárból



Tömbök

- Már bemutattuk a tömbök használatát
- Miért jobb a tömb a többi konténernél?
  - Hatékonyság: konstans idejű elérés, de fix a méret (flexibilis tömb: `ArrayList` konténer)
  - Típus: nem veszíti el a típust – adott típusú elemeket tárol, nem `Object`-eket (ezt már a generikus konténerek is tudják)
- Először mindig próbáljunk meg tömböt használni, és csak akkor válasszunk másik konténert, ha korlátba ütközünk

Az `Arrays` osztály

- A `java.util` könyvtár része
- Statikus metódusokat tartalmaz, amik segítséget nyújtanak a tömbkezelésben:
  - `equals()` – tömbök (érték szerinti) egyenlősége
  - `fill()` – tömb feltöltése adott értékkel/referenciával
  - `sort()` – tömb rendezése
  - `binarySearch()` – tömbben keresés
- Mindegyik meg van valósítva (overload) minden primitív típusra és az `Object`-re

Tömbök

Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```

Tömbök

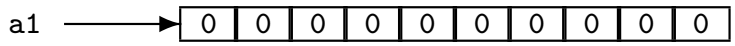
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

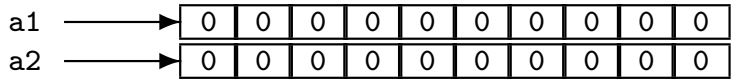
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

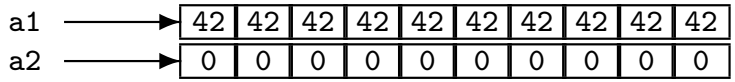
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

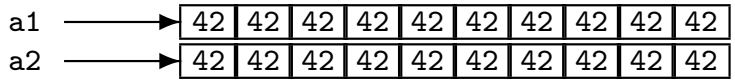
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

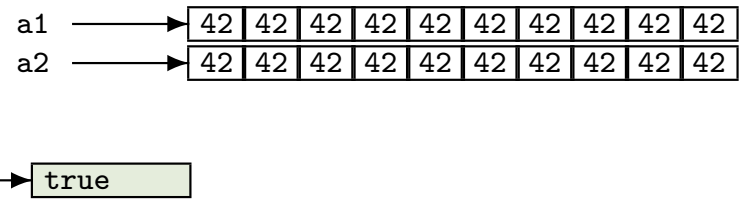
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

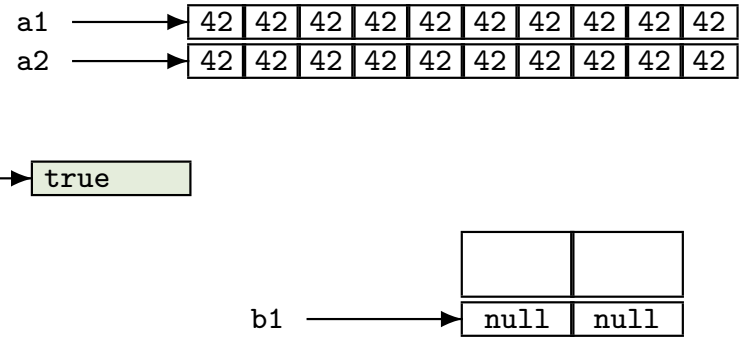
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

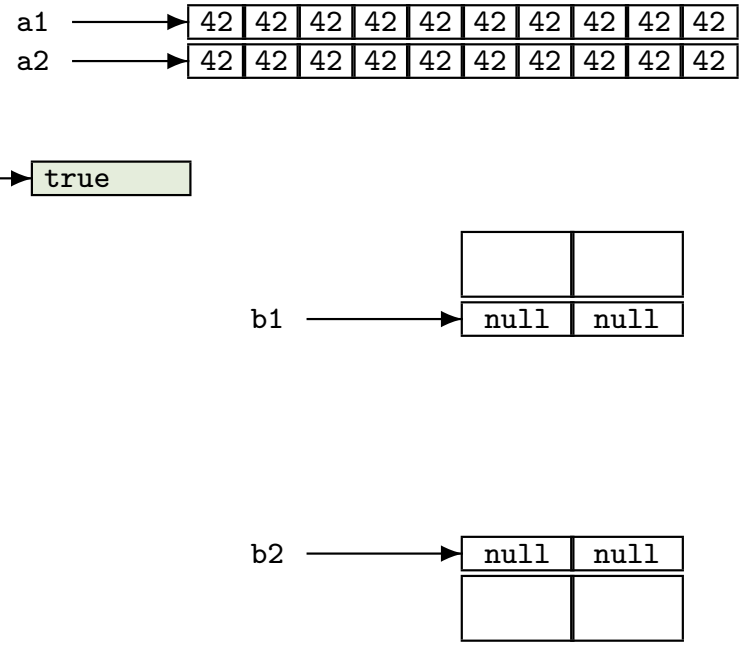
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

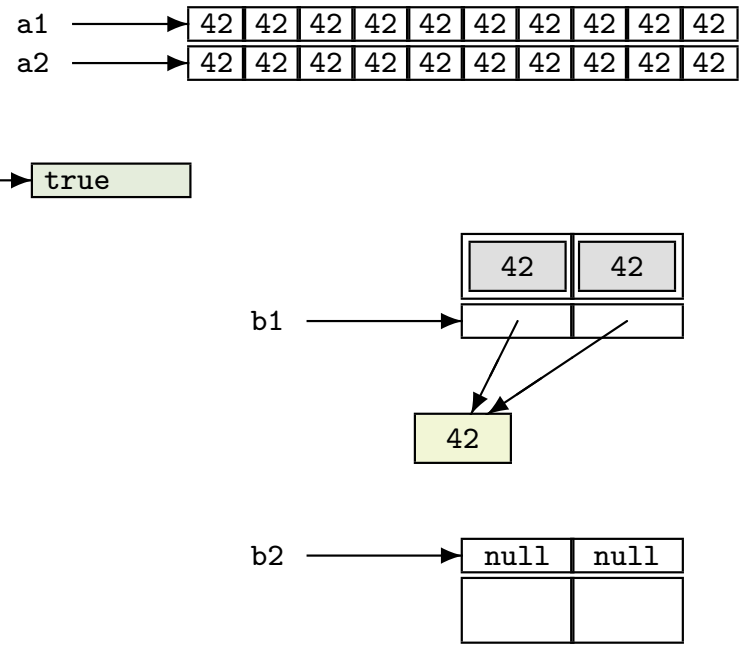
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

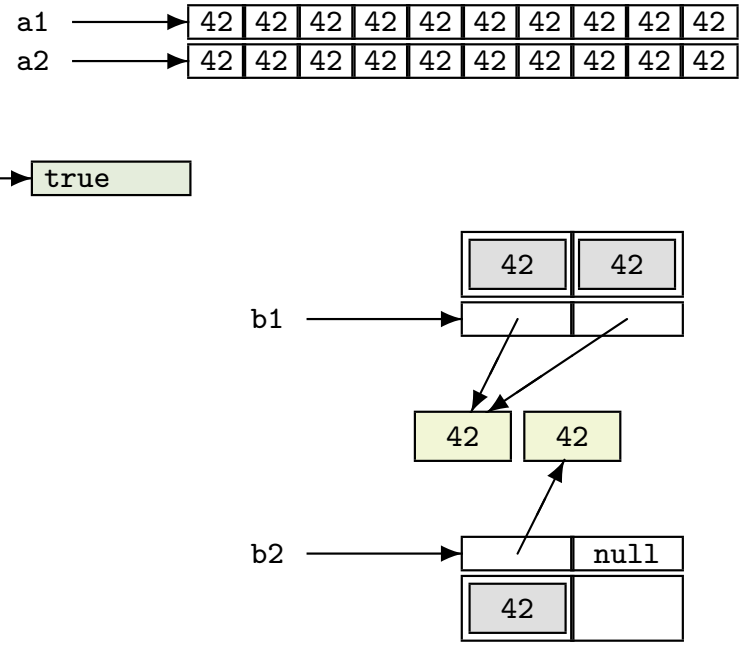
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

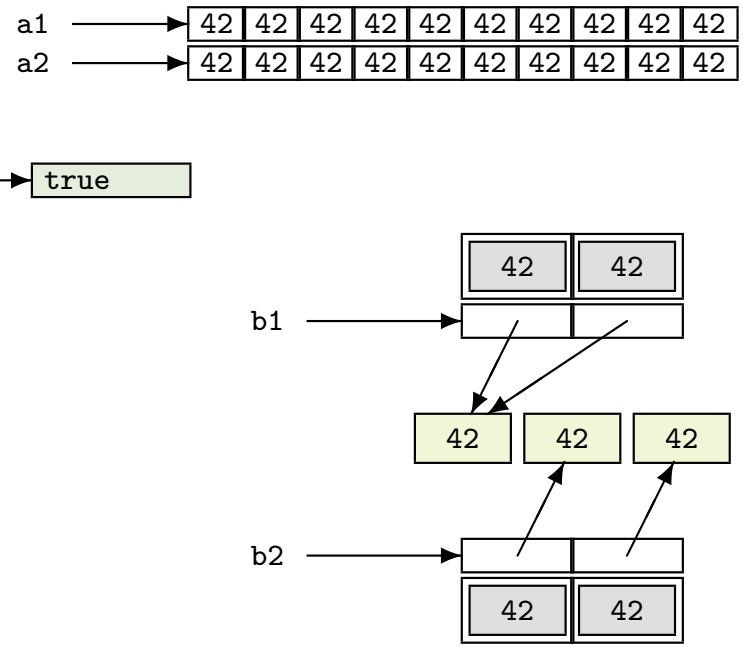
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

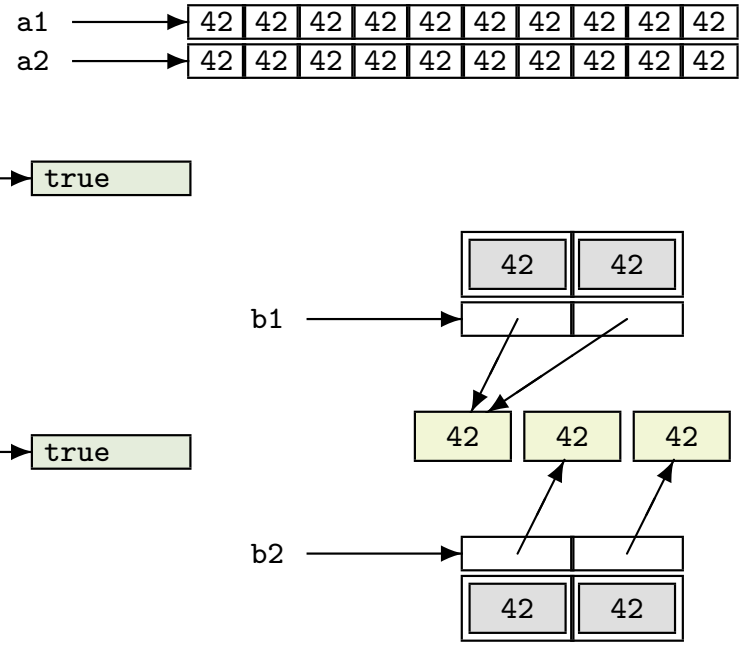
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

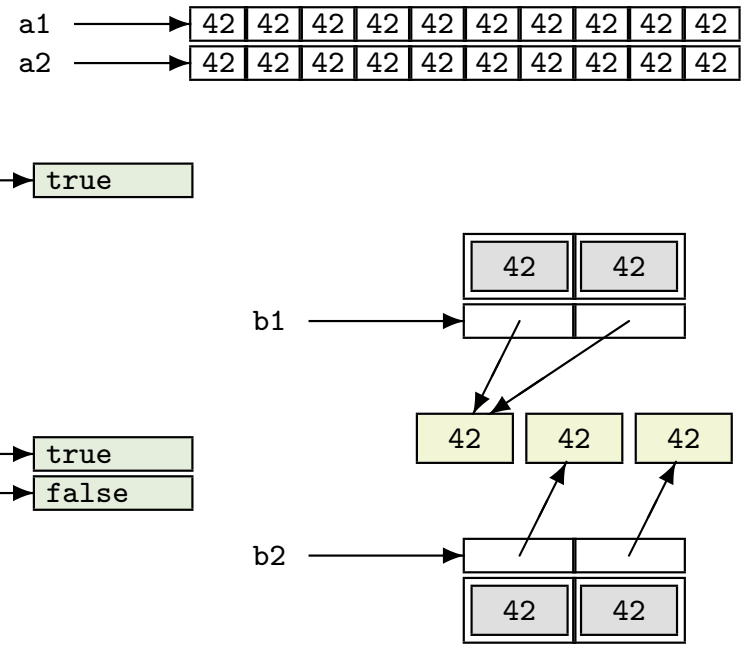
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

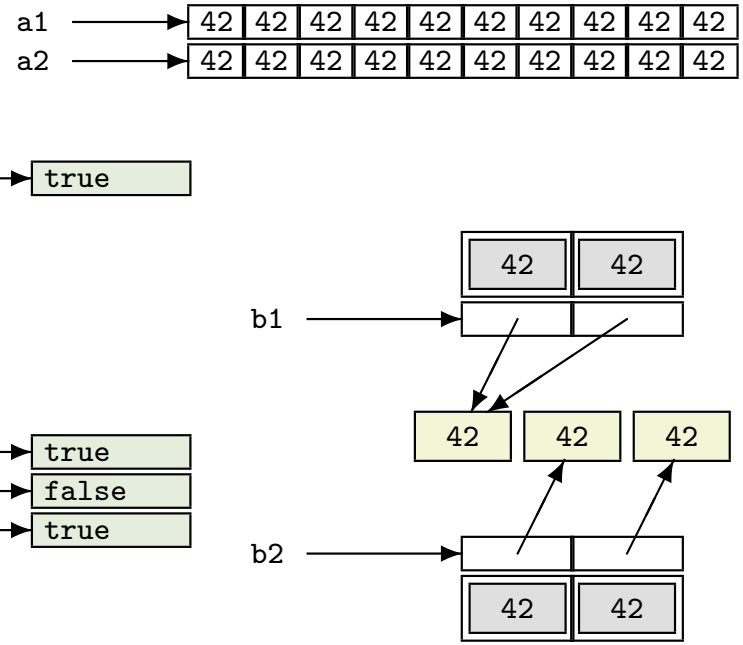
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

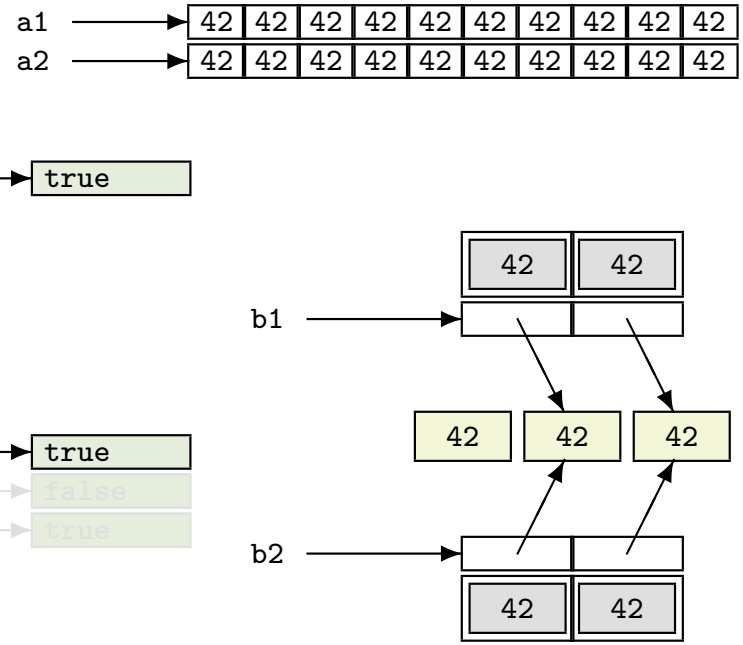
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

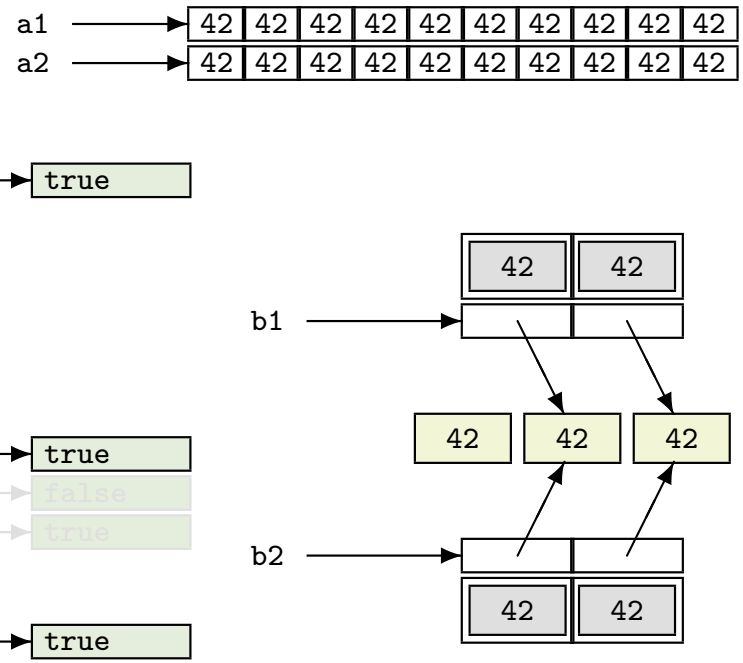
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök

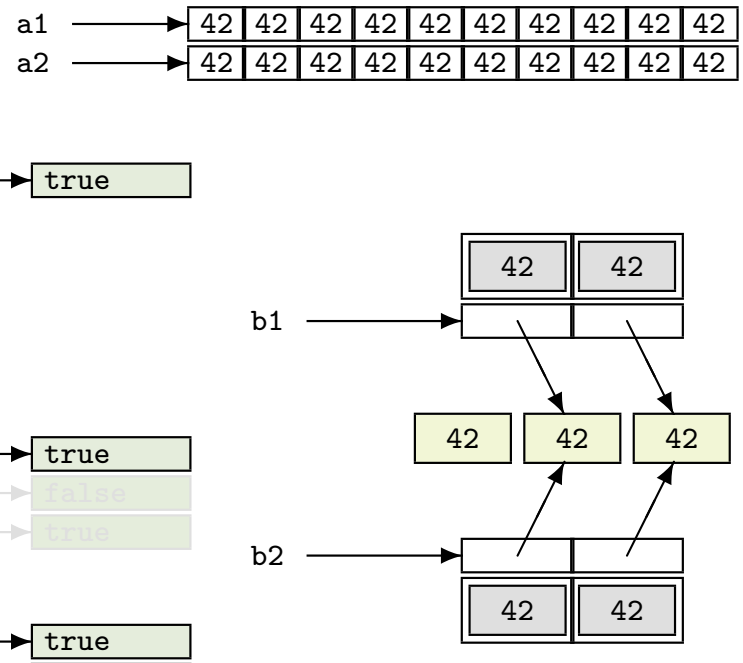
Egyenlőség, feltöltés, másolás

```
import java.util.*;

public class TombPelda {
    public static void main(String[] args) {
        int[] a1 = new int[10];
        int[] a2 = new int[10];
        Arrays.fill(a1, 42);
        Arrays.fill(a2, 42);
        System.out.println(Arrays.equals(a1,a2));

        Integer[] b1 = new Integer[2];
        Integer[] b2 = new Integer[2];
        Arrays.fill(b1, new Integer(42));
        b2[0] = new Integer(42);
        b2[1] = new Integer(42);
        System.out.println(Arrays.equals(b1,b2));
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);

        System.arraycopy(b2, 0, b1, 0, b2.length);
        System.out.println(b1[0] == b2[0]);
        System.out.println(b1[0] == b1[1]);
    }
}
```



Tömbök rendezése

- Az általános rendezőalgoritmusoknak
  - rendezés közben össze kell tudni hasonlítani két elemet, de
  - a rendezés csak felhasználja az összehasonlítás eredményét, nem ő dönt, és
  - a mechanizmusnak típusoktól függetlenül kell működnie.
- Ez *callback* technikával oldható meg
  - a rendezés „megkér” valakit az összehasonlításra, ahelyett, hogy ő maga végezné el
  - két módszer van: `Comparable` és `Comparator`

Tömbök rendezése

Comparable

- A „természetes” módszer: a rendezendő objektumok maguk el tudják dönteni, hogy egy másik objektumhoz hogyan viszonyulnak
- A `java.lang.Comparable` interfészt kell a rendezendő objektumok osztálynak implementálnia
- Ennek egyetlen metódusa van: `compareTo(Object)`
  - Egy `Object` típusú argumentumot vár paraméterül.
  - A visszatérési értéke
    - <0, ha a „megkérdezett” objektum kisebb mint az argumentum,
    - 0, ha az objektum és az argumentum egyenlő, és
    - >0, ha az objektum nagyobb, mint az argumentum.

Tömbök rendezése

Comparable példa

```
import java.util.*;

class Elem implements Comparable {
    double i;
    public Elem(double n) {
        i = n;
    }
    public int compareTo(Object o) {
        double oi = ((Elem)o).i;
        return (i < oi ? -1 : (i == oi ? 0 : 1));
    }
    public String toString() {
        return new Double(i).toString();
    }
}
```

```
public class ComparablePelda {
    static void print(Elem[] t) {
        for (int i = 0; i < t.length; ++i)
            System.out.println(t[i]);
        System.out.println();
    }
    public static void main(String[] args) {
        Elem[] t = new Elem[10];
        for (int i = 0; i < t.length; ++i)
            t[i] = new Elem(Math.random());
        print(t);
        Arrays.sort(t);
        print(t);
    }
}
```

```
0.36115112579542097
0.5312228660684087
0.6616367146790236
0.4806892330436644
0.2803566373024132
0.33038515437895566
0.4804521551226102
0.04078726950214795
0.8404921623838968
0.16426939999947487

0.04078726950214795
0.16426939999947487
0.2803566373024132
0.33038515437895566
0.36115112579542097
0.4804521551226102
0.4806892330436644
0.5312228660684087
0.6616367146790236
0.8404921623838968
```

Tömbök rendezése

Comparator

- A „másik” módszer: egy külön objektum dönteni el, hogy két objektum hogyan viszonyul egymáshoz
- A `java.lang.Comparator` interfészt kell a rendező objektum osztálynak implementálnia
- Ennek egyetlen metódusa van: `compare(Object,Object)`
  - Két `Object` típusú argumentumot vár paraméterül.
  - A visszatérési értéke
    - <0, ha az első argumentum kisebb mint a második,
    - 0, ha a két argumentum egyenlő, és
    - >0, ha az első argumentum nagyobb, mint a második.

Tömbök rendezése

Comparator példa

```
import java.util.*;

class Elem {
    double i;
    public Elem(double n) {
        i = n;
    }
    public String toString() {
        return new Double(i).toString();
    }
}
```

```
class ElemComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        double i1 = ((Elem)o1).i;
        double i2 = ((Elem)o2).i;
        return (i1 <  i2 ? -1 : (i1 == i2 ? 0 : 1));
    }
}
```

```
public class ComparatorPelda {
    static void print(Elem[] t) {
        for (int i = 0; i < t.length; ++i)
            System.out.println(t[i]);
        System.out.println();
    }
    public static void main(String[] args) {
        Elem[] t = new Elem[10];
        for (int i = 0; i < t.length; ++i)
            t[i] = new Elem(Math.random());
        print(t);
        Arrays.sort(t, new ElemComparator());
        print(t);
    }
}
```

```
0.23270579517348267
0.6584103370523003
0.7083382482863732
0.8784676009733516
0.9408362641908479
0.4261445649240052
0.05242737386595098
0.7351894876299406
0.09346485500758372
0.42212119404147186

0.05242737386595098
0.09346485500758372
0.23270579517348267
0.42212119404147186
0.4261445649240052
0.6584103370523003
0.7083382482863732
0.7351894876299406
0.8784676009733516
0.9408362641908479
```

Tömbben keresés

- Csak rendezett tömbben lehet keresni!
- Az `Arrays.binarySearch()` metódus bináris keresést valósít meg.
  - Visszatérési értéke
    - a keresett elem indexe (≥ 0), ha az elem benne van a tömbben,
    - $-idx - 1$ (<0), ahol idx annak az elemnek az indexe, amelyik előtt a keresett elemnek lennie kellene
  - Ha több elem is megfelel a keresésnek, akkor bizonytalan, hogy melyiket találja meg
  - Ha `Comparator`-ral lett rendezve, akkor ugyanazt a `Comparator` objektumot kell megadni a keresésnek is, mint a rendezésnek

Tömbben keresés

Példa

```
public class TombbenKereses {
    public static void main(String[] args) {
        int[] t = new int[100];
        /* tömb feltöltése */
        Arrays.sort(t);
        /* ... */
        int pozicio = Arrays.binarySearch(t,19);
        if (pozicio >= 0) {
            /* megvan */
        }
    }
}
```

Tartalom

Kollekciók (konténerek)

- A konténer osztálykönyvtár egyike a leghatékonyabb programozási eszközöknek
- Java 2-ben teljesen újratervezték
  - az első Java verziókban nagyon gyengére sikeredett
- Java 5-ben kibővítették generikus típusokra
- Két koncepció
  - *Kollekció* (`Collection`): Egyéni elemek csoportja, általában valamilyen szabályossággal (listában sorrendiség, halmazban egyediség, stb.)
  - *Leképezés* (`Map`): Kulcs-adat kettősök csoportja, a kulcsra gyors keresést biztosít

Kollekciók

Példa

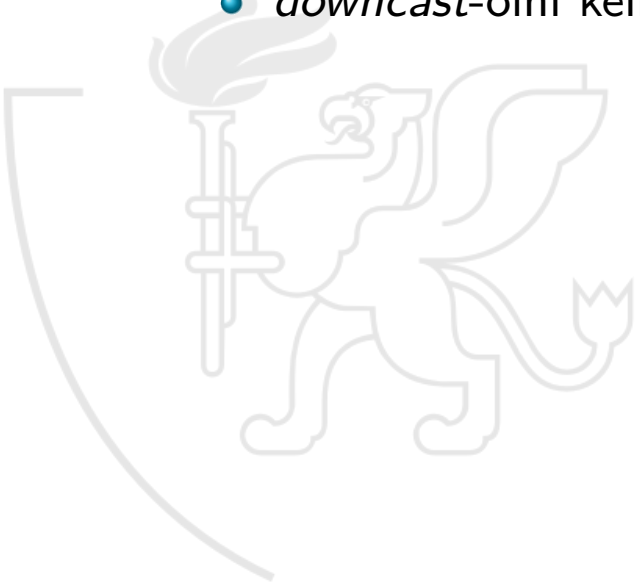
```
import java.util.*;

public class Kollekciok {
    static Collection feltolt(Collection c) {
        c.add("kutya");
        c.add("macska");
        c.add("macska");
        return c;
    }
    static Map feltolt(Map m) {
        m.put("kutya", "Ubul");
        m.put("macska", "Arlene");
        m.put("macska", "Garfield");
        return m;
    }
    public static void main(String[] args) {
        System.out.println(feltolt(new ArrayList()));
        System.out.println(feltolt(new HashSet()));
        System.out.println(feltolt(new HashMap()));
    }
}
```

```
[kutya, macska, macska]
[kutya, macska]
{kutya=Ubul, macska=Garfield}
```

A Java 5 előtti kollekciók

- Elveszítik a típust
  - mindig `Object`-eket tárolnak
  - általános célúak
    - nem tárolhatnak specifikus típusú objektumokat
  - Bármi belerakható – nincs típusellenőrzés
    - pl. egy kutyákat tároló konténerbe nyugodtan bele lehet tenni macskákat
    - `Object` referencia kerül bele (*upcast*)
    - *downcast*-olni kell az elemet használat előtt!



A Java 5 előtti kollekciók

Példa

```
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```
import java.util.*;

public class KutyakEsMacskak {
    public static void main(String[] args) {
        ArrayList kutyak = new ArrayList();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        kutyak.add(new Macska(3));
        for(int i = 0; i < kutyak.size(); i++)
            ((Kutya)kutyak.get(i)).print();
    }
}
```

```
Kutya #0
Kutya #1
Kutya #2
Exception in thread "main" java.lang.ClassCastException
at KutyakEsMacskak.main(KutyakEsMacskak.java:10)
```

Iterátorok

- Egy iterátor egy objektum, amely végighalad egy objektumsorozaton anélkül, hogy a felhasználó programozó tudná, hogy milyen konkrét belső struktúrája van a kollekciónak
- Java iterátor használata
  - Minden konténertől kérni lehet egy `Iterator` típusú objektumot az `iterator()` metódushívással
  - Ettől az `Iterator` típusú objektumtól a „következő” `Object` típusú elemet a `next()`-tel lehet kérni (a legelsőt is így kérjük!)
  - Az `Iterator` objektum `hasNext()` metódusa megmondja, hogy van-e következő elem
  - Az `Iterator` objektumon keresztül lekért utplsó elemet az iterátor `remove()` metódusával lehet törölni

Iterátorok

Példa

```
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```
import java.util.*;

public class KutyakEsMacskakIterator {
    public static void main(String[] args) {
        ArrayList kutyak = new ArrayList();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        Iterator i = kutyak.iterator();
        while(i.hasNext())
            ((Kutya)(i.next())).print();
    }
}
```

```
Kutya #0
Kutya #1
Kutya #2
```

Foreach

Példa

```
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```
import java.util.*;

public class KutyakEsMacskakForeach {
    public static void main(String[] args) {
        ArrayList kutyak = new ArrayList();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        for(Object o : kutyak)
            ((Kutya)o).print();
    }
}
```

```
Kutya #0
Kutya #1
Kutya #2
```

Lista (Collection)

- `List` interfész
  - Elemeket tárol **adott sorrendben**
  - Speciális iterátor: `ListIterator`, amely figyel a sorrendre
- `ArrayList`
  - Tömbbel megvalósított lista
  - Gyors elérés de lassú beszúrás/törlés
- `LinkedList`
  - „Igazi” láncolt lista
  - Gyors beszúrás/törlés de lassú elérés
  - Plusz metódusok
    - `addFirst()`, `addLast()`
    - `getFirst()`, `getLast()`
    - `removeFirst()`, `removeLast()`

Halmaz (Collection)

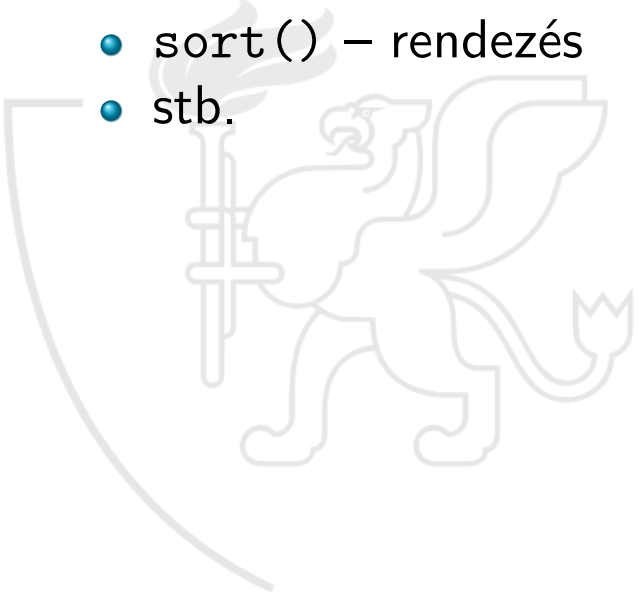
- `Set` interfész
  - **Egyedi elemeket** tárol sorrendiség nélkül
  - A tárolt elemeknek felül kell definiálni az `equals()` és `hashCode()` metódusát
- `HashSet`
  - Gyors keresést biztosító rendezetlen halmaz
- `TreeSet`
  - Rendezett halmaz (fával megvalósítva)
  - Rendezhető elemek (`Comparable`) vagy külön rendező objektum (`Comparator`) szükséges
  - Plusz metódusok
    - `first()` – a legkisebb elem
    - `last()` – a legnagyobb elem

Leképezés (Map)

- `Map` interfész
  - Kulcs-adat objektum-párok csoportja
  - A kulcsra gyors keresést biztosít
- `HashMap`
  - A Hash-táblával implementálva
  - Keresésre/beszúrásra optimalizálva
- `TreeMap`
  - Piros-fekete fával implementálva
  - Rendezetten tárolja az elemeket
  - Rendezhető kulcs (`Comparable`) vagy külön rendező objektum (`Comparator`) szükséges
  - Plusz metódusok
    - `firstKey()` – a legkisebb kulcs
    - `lastKey()` – a legnagyobb kulcs

A `Collections` osztály

- A `java.util` csomag része
- Statikus metódusokat tartalmaz, amelyek segítséget nyújtanak a kollekciók kezelésében (hasonlóan, mint az `Arrays` osztály a tömbök esetében)
  - `binarySearch()` – keresés
  - `copy()` – másolás
  - `fill()` – feltöltés (csere) adott értékkel
  - `reverse()` – elemsorrend megfordítás
  - `sort()` – rendezés
  - stb.



Tartalom

Generikus kollekciók (Java 5-től)

- Megtartják a típust
  - biztonságosabbak
  - speciális célúak
    - specifikus típusú objektumokat tárolnak
  - Csak a megfelelő típus rakható be – típusellenőrzés fordításkor
    - pl. egy kutyákat tároló konténerbe nem lehet macskákat beletenni
    - adott típusú referencia kerül bele (nincs *upcast*)
    - nem kell *downcast*-olni az elemet használat előtt

Kollekciók

Generikus példa

```java
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```java
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```java
import java.util.*;

public class KutyakEsMacskakGen {
    public static void main(String[] args) {
        ArrayList<Kutya> kutyak = new ArrayList<Kutya>();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        kutyak.add(new Macska(3));  // fordítási hiba
        for(int i = 0; i < kutyak.size(); i++)
            kutyak.get(i).print();
    }
}
```

```
KutyakEsMacskakGen.java:8: cannot find symbol
symbol : method add(Macska)
location: class java.util.ArrayList<Kutya>
kutyak.add(new Macska(3));  // fordítási hiba
^
```

Iterátorok

Generikus példa

```
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```
import java.util.*;

public class KutyakEsMacskakGenIterator {
    public static void main(String[] args) {
        ArrayList<Kutya> kutyak = new ArrayList<Kutya>();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        Iterator<Kutya> i = kutyak.iterator();
        while(i.hasNext())
            i.next().print();
    }
}
```

```
Kutya #0
Kutya #1
Kutya #2
```

Foreach

Generikus példa

```java
public class Kutya {
    private int kutyaSz;
    Kutya(int i) {
        kutyaSz = i;
    }
    public void print() {
        System.out.println("Kutya #" + kutyaSz);
    }
}
```

```java
public class Macska {
    private int macskaSz;
    Macska(int i) {
        macskaSz = i;
    }
    public void print() {
        System.out.println("Macska #" + macskaSz);
    }
}
```

```java
import java.util.*;

public class KutyakEsMacskakGenForeach {
    public static void main(String[] args) {
        ArrayList<Kutya> kutyak = new ArrayList<Kutya>();
        for(int i = 0; i < 3; i++)
            kutyak.add(new Kutya(i));
        for(Kutya k : kutyak)
            k.print();
    }
}
```

```
Kutya #0
Kutya #1
Kutya #2
```

Tömbök rendezése

Generikus Comparable példa

```
import java.util.*;

class Elem implements Comparable<Elem> {
    double i;
    public Elem(double n) {
        i = n;
    }
    public int compareTo(Elem e) {
        double ei = e.i;
        return (i < ei ? -1 : (i == ei ? 0 : 1));
    }
    public String toString() {
        return new Double(i).toString();
    }
}
```

```
public class ComparableGenPelda {
    static void print(Elem[] t) {
        for (int i = 0; i < t.length; ++i)
            System.out.println(t[i]);
        System.out.println();
    }
    public static void main(String[] args) {
        Elem[] t = new Elem[10];
        for (int i = 0; i < t.length; ++i)
            t[i] = new Elem(Math.random());
        print(t);
        Arrays.sort(t);
        print(t);
    }
}
```

```
0.36115112579542097
0.5312228660684087
0.6616367146790236
0.4806892330436644
0.2803566373024132
0.33038515437895566
0.4804521551226102
0.04078726950214795
0.8404921623838968
0.16426939999947487

0.04078726950214795
0.16426939999947487
0.2803566373024132
0.33038515437895566
0.36115112579542097
0.4804521551226102
0.4806892330436644
0.5312228660684087
0.6616367146790236
0.8404921623838968
```

Tömbök rendezése

Generikus Comparator példa

```
import java.util.*;

class Elem {
    double i;
    public Elem(double n) {
        i = n;
    }
    public String toString() {
        return new Double(i).toString();
    }
}
```

```
class ElemComparator implements Comparator<Elem> {
    public int compare(Elem o1, Elem o2) {
        double i1 = o1.i;
        double i2 = o2.i;
        return (i1 <  i2 ? -1 : (i1 == i2 ? 0 : 1));
    }
}
```

```
public class ComparatorGenPelda {
    static void print(Elem[] t) {
        for (int i = 0; i < t.length; ++i)
            System.out.println(t[i]);
        System.out.println();
    }
    public static void main(String[] args) {
        Elem[] t = new Elem[10];
        for (int i = 0; i < t.length; ++i)
            t[i] = new Elem(Math.random());
        print(t);
        Arrays.sort(t, new ElemComparator());
        print(t);
    }
}
```

```
0.23270579517348267
0.6584103370523003
0.7083382482863732
0.8784676009733516
0.9408362641908479
0.4261445649240052
0.05242737386595098
0.7351894876299406
0.09346485500758372
0.42212119404147186

0.05242737386595098
0.09346485500758372
0.23270579517348267
0.42212119404147186
0.4261445649240052
0.6584103370523003
0.7083382482863732
0.7351894876299406
0.8784676009733516
0.9408362641908479
```