Programozási nyelvek Java

Típusok

Kozsik Tamás

ELTE
IK

Öröklődés $\Rightarrow$ altípusosság

`class A implements I`

$\mathsf{A}\ \Delta_{ci}\ \mathsf{I} \Rightarrow \mathsf{A} <: \mathsf{I}$

`class A extends B`

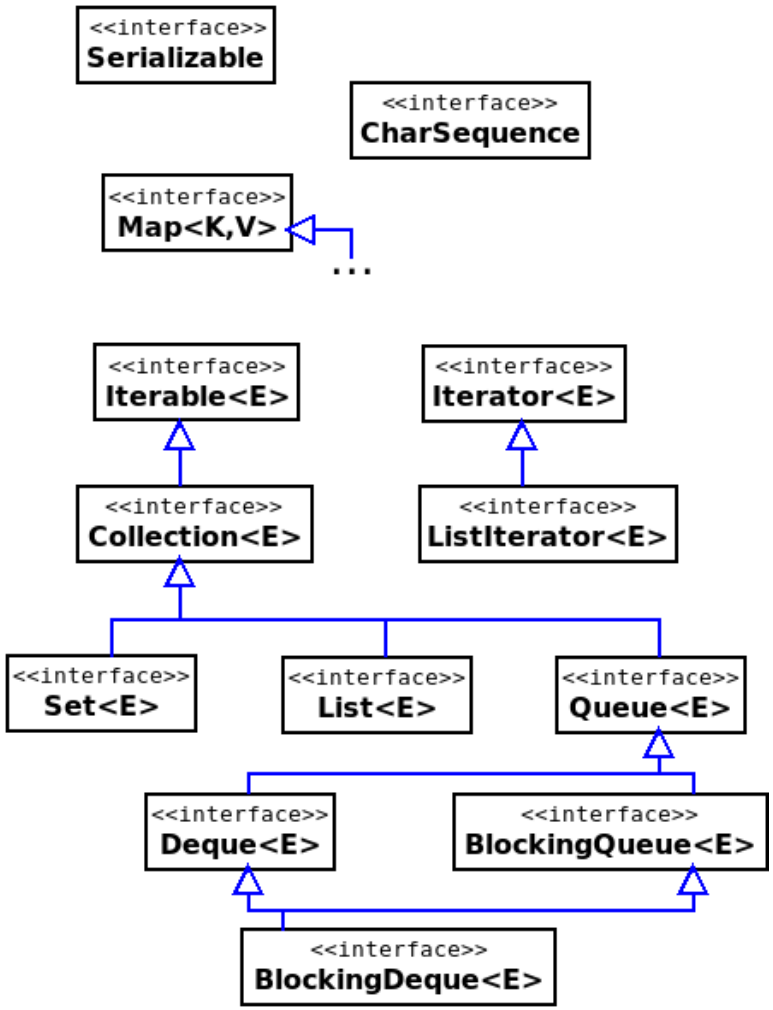
$\mathsf{A}\ \Delta_{c}\ \mathsf{B} \Rightarrow \mathsf{A} <: \mathsf{B}$

`interface I extends J`

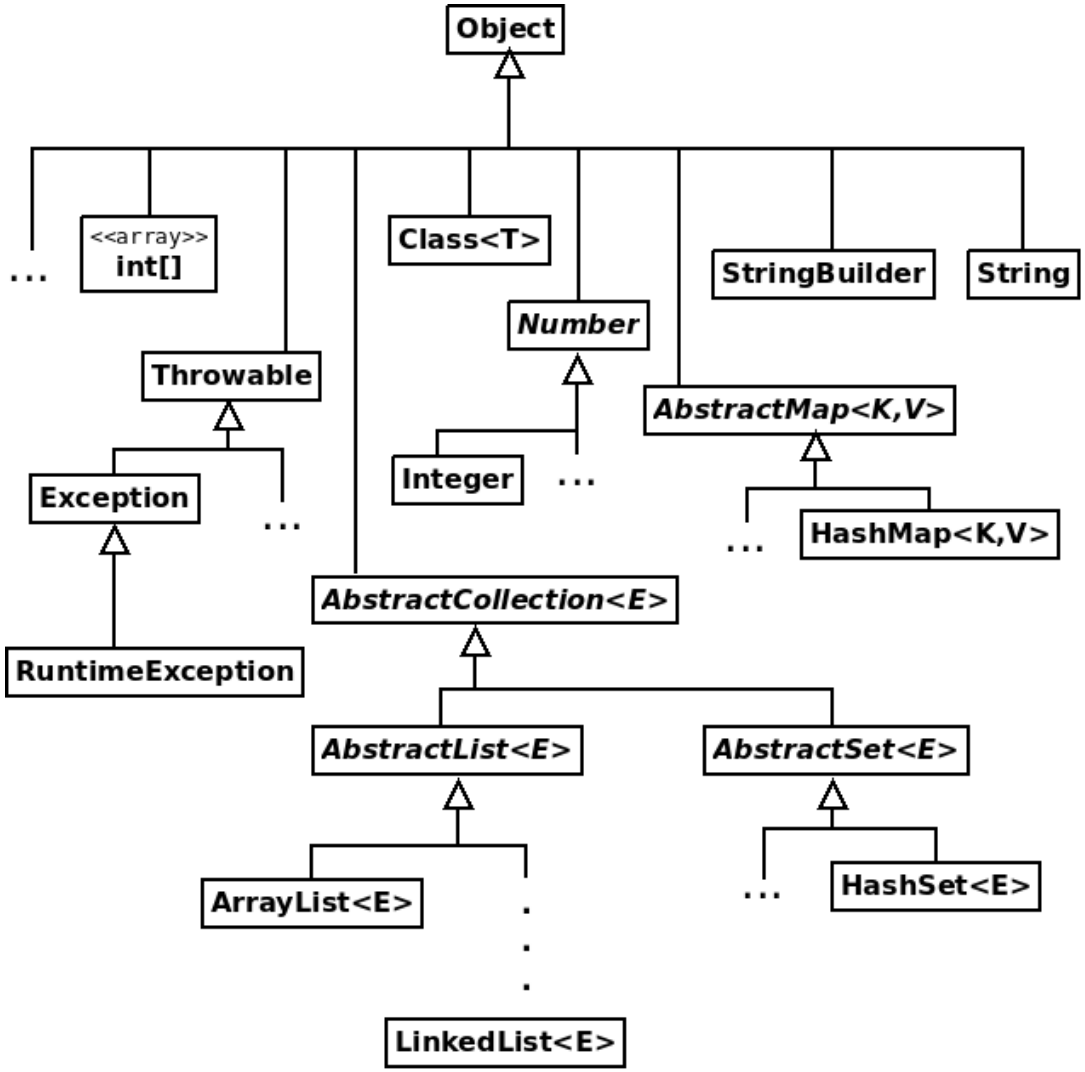
$\mathsf{I}\ \Delta_{i}\ \mathsf{J} \Rightarrow \mathsf{I} <: \mathsf{J}$

ELTE
IK

Interface-ek hierarchiája a Javában (részlet)



Osztályok hierarchiája a Javában (részlet)



java.lang.Object

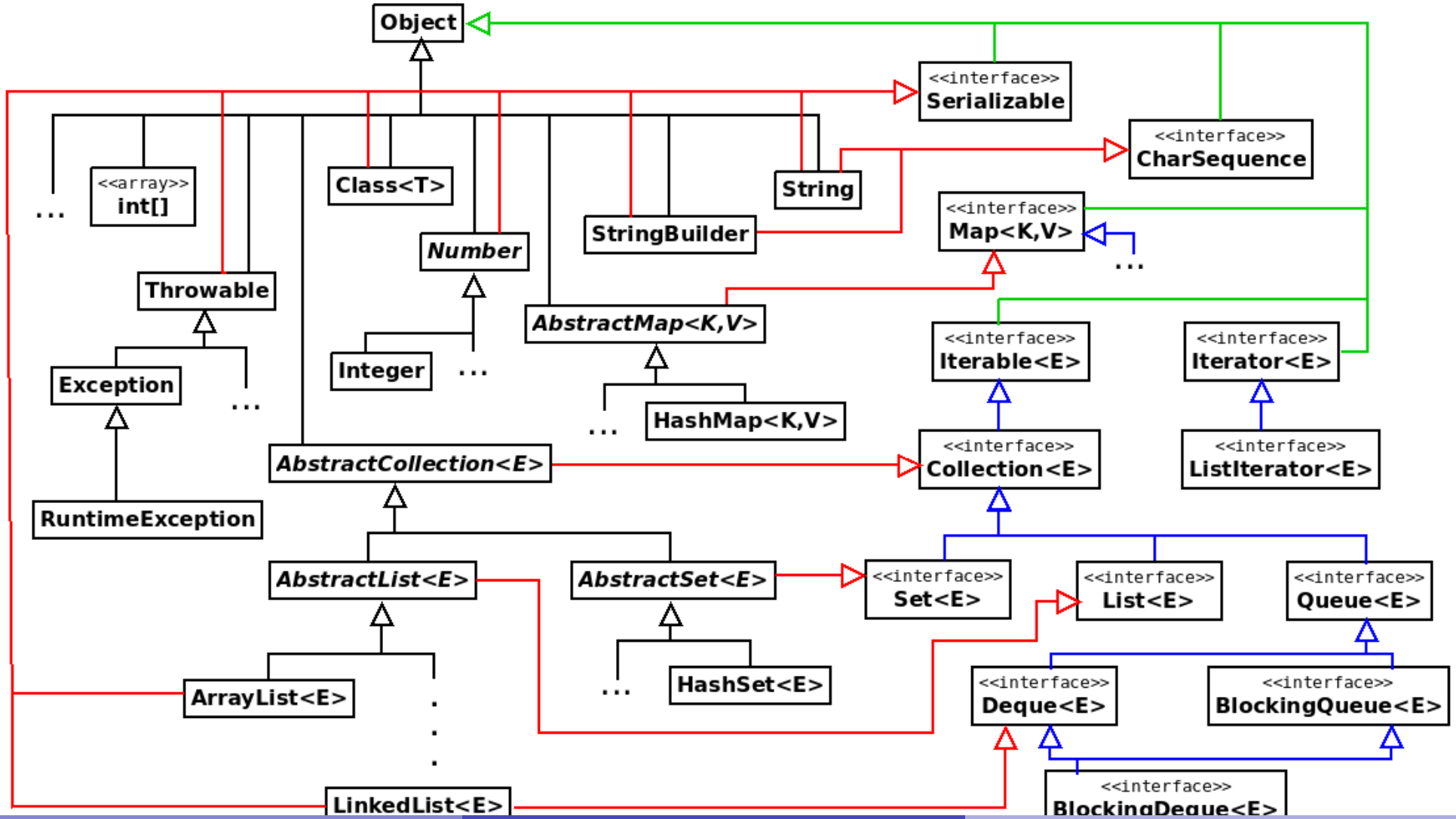
Minden osztály belőle származik, kivéve önmagát!

```
package java.lang;
public class Object {
    public Object() { ... }
    public String toString() { ... }
    public int hashCode() { ... }
    public boolean equals(Object that) { ... }
    public Class getClass() { ... }
    ...
}
```

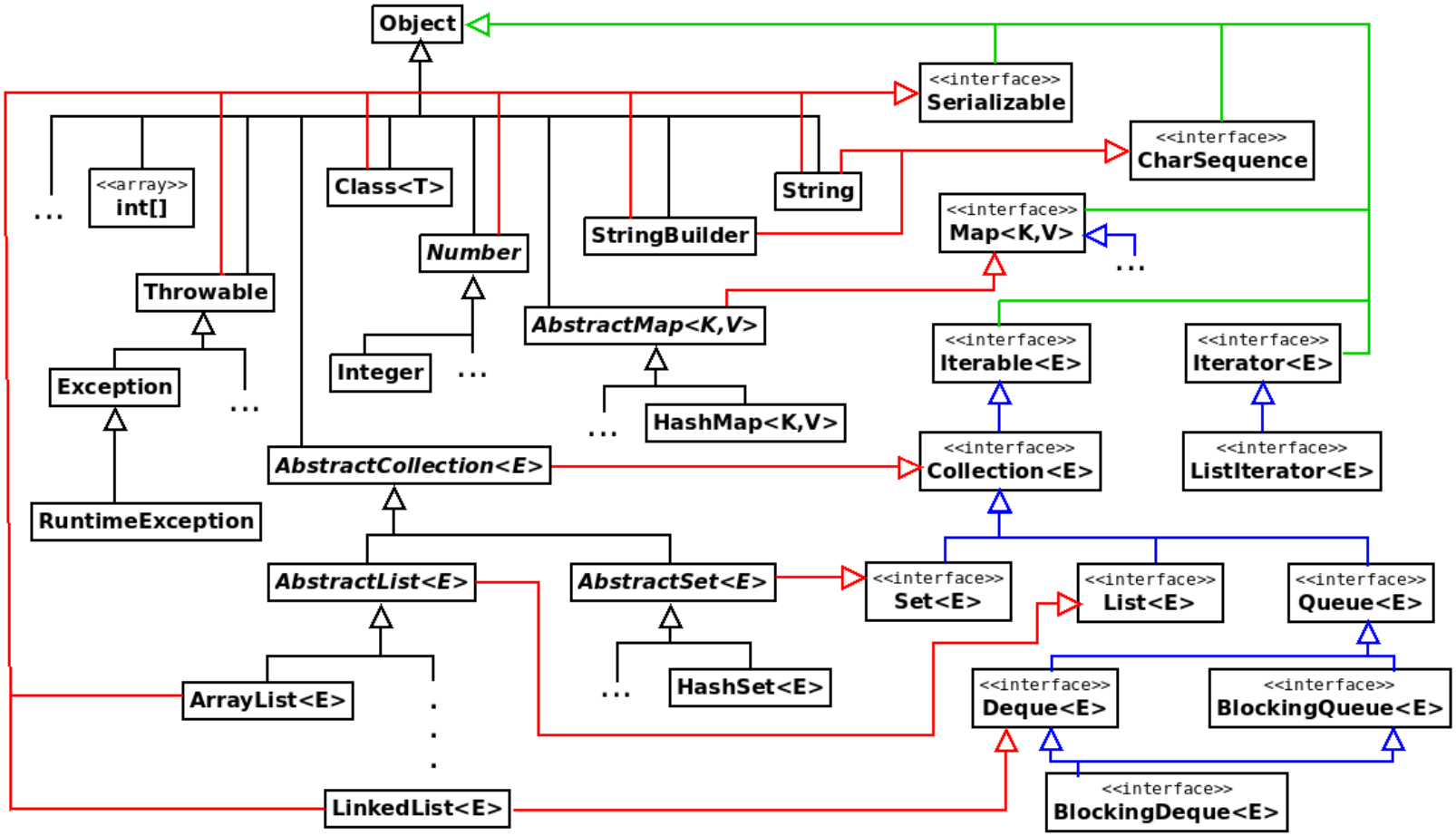
ELTE
IK

Referenciatípusok hierarchiája a Javában (részlet)

körmentes irányított gráf (DAG: directed acyclic graph)



Típusok hierarchiája a Javában (részlet)




```
boolean
char
byte
short
int
long
float
double
```

Kivételosztályok hierarchiája

‘java.lang.Throwable‘

- ‘java.lang.Exception‘
  - ‘java.sql.SQLException‘
  - ‘java.io.IOException‘
    - ‘java.io.FileNotFoundException‘
  - ...
  - saját kivételek általában ide kerülnek
  - ‘java.lang.RuntimeException‘
    - ‘java.lang.NullPointerException‘
    - ‘java.lang.ArrayIndexOutOfBoundsException‘
    - ‘java.lang.IllegalArgumentException‘
    - ...
- ‘java.lang.Error‘
  - ‘java.lang.VirtualMachineError‘
  - ...

Nem ellenőrzött kivételek

- `java.lang.RuntimeException` és leszármazottjai
- `java.lang.Error` és leszármazottjai

Egyes alkalmazási területen akár ezek is kezelendők!

Kivételkezelő ágak

```
try {
    ...
} catch (FileNotFoundException e) {
    ...
} catch (EOFException e) {
    ...
} // nem kezeltük a java.net.SocketException-t
```

Speciálisabb-általánosabb kivételkezelő ágak

```
try {
    ...
} catch (FileNotFoundException e) {
    ...
} catch (EOFException e) {
    ...
} catch (IOException e) {  // minden egyéb IOException
    ...
}
```

ELTE
IK

Fordítási hiba: elérhetetlen kód

```
try {
    ...
} catch (FileNotFoundException e) {
    ...
} catch (IOException e) {  // minden egyéb IOException
    ...
} catch (EOFException e) { // rossz sorrend!
    ...
}
```

ELTE
IK

Altípus reláció

$$<: \quad = \quad (\Delta_c \cup \Delta_i \cup \Delta_{ci} \cup \Delta_o)^*$$

- Δ_o jelentése: minden a `java.lang.Object`-ből származik
- ϱ^* jelentése: ϱ reláció reflexív, tranzitív lezártja
  - ◇ Ha `A` ϱ `B`, akkor `A` ϱ^* `B`
  - ◇ Reflexív lezárt: `A` ϱ^* `A`
  - ◇ Tranzitív lezárt: ha `A` ϱ^* `B` és `B` ϱ^* `C`, akkor `A` ϱ^* `C`

Ez egy parciális rendezés (RAT)!

A dinamikus típus a statikus típus altípusa

Ha `A` $<:$ `B`, akkor

- `B v = new A();` helyes
- `void m(B p)...` esetén `m(new A())` helyes
- `A a; B b; ... b = a;` helyes

Altípusos polimorfizmus (subtype polymorphism)

Ha egy kódbázist megírtunk, újrahasznosíthatjuk speciális típusokra!

- Általánosabb típusok helyett használhatunk altípusokat
- Több típusra is működik a kódbázis: polimorfizmus

Újrafelhasználhatóság!

Specializálás

- Az altípus „mindent tud”, amit a bázistípus
- Az altípus speciálisabb lehet
- Ez az *is-egy* reláció
  - ◇ `Car` *is-a* `Vehicle`
  - ◇ `Boat` *is-a* `Vehicle`
- Emberi gondolkodás, OO modellezés

ELTE IK

Többszörös altípusképzés

- Egy fogalom több általános fogalom alá tartozhat
  - ◇ `PoliceCar` *is-a* `Car` **és** *is-a* `EmergencyVehicle`
  - ◇ `FireBoat` *is-a* `Boat` **és** *is-a* `EmergencyVehicle`
- Összetett fogalmi modellezés Javában: `interface`

ELTE
IK

Többszörös kódöröklés?

- Kódöröklés: osztályok mentén
  - ◇ csak egyszeres öröklődés

Többszörös kódöröklés?

- Kódöröklés: osztályok mentén
  - csak egyszeres öröklődés
- `interface`-ekből
  - többszörös öröklődés
  - Korlátozott mértékű kódöröklés: `default` implementációjú példánymetódusok

Speciális jelentésű interfészek

```
class DataStructure<T> implements java.lang.Iterable<T>
// működik rá a bejáró ciklus

class Resource implements java.lang.AutoCloseable
// működik rá a try-with-resources

class Rational implements java.lang.Cloneable
// működik rá a (sekély) másolás

class Data implements java.io.Serializable
// működik rá az objektumszerializáció
```

ELTE
IK

Iterable és Iterator

- `Iterator` (bejáró): sorban érinti egy adatszerkezet elemeit
- `Iterable` (bejárható): adatszerkezet, amitől kérhetünk bejárót

java.util.Iterator

```
public interface Iterator<T> {
  boolean hasNext();
  T next();
  ...
}
```

java.lang.Iterable

```
public interface Iterable<T> {
  java.util.Iterator<T> iterator();
  ...
}
```

Iterator elképzelt megvalósítása

```
package java.util;
public class ArrayList<T> implements Iterable<T> {
  Object[] data;
  int size = 0;
  ...
  public Iterator<T> iterator() { return new ALIterator<>(this
}
class ALIterator<T> implements Iterator<T> {
  private final ArrayList<T> theArrayList;
  private int index = 0;
  ALIterator(ArrayList<T> al) { theArrayList = al; }
  public boolean hasNext() { return index < theArrayList.size
  @SuppressWarnings("unchecked") public T next() {
    return (T)theArrayList.data[index++];
  }
}
```

Iterable és Iterator – polimorfizmus

```
long sum(Iterable<Integer> is) {
    long sum = 0L;
    Iterator<Integer> it = is.iterator();
    while (it.hasNext()) {
        sum += it.next();
    }
    return sum;
}
```

```
List<Integer> list = new LinkedList<>();
...
long sum = sum(list);
```

IK

Bejáró ciklus

```
long sum(Iterable<Integer> is) {
  long sum = 0L;

  for (Integer item: is) {
    sum += item;
  }
  return sum;
}
```

```
List<Integer> list = new LinkedList<>();
...
long sum = sum(list);
```

IK

Többszörös bejárás

```
List<Pair<Integer,Integer>> pairs(List<Integer> ns) {
  List<Pair<Integer,Integer>> ps = new LinkedList<>();
  Iterator<Integer> it = ns.iterator();
  while (it.hasNext()) {
    Integer item = it.next();
    Iterator<Integer> it2 = ns.iterator();
    while (it2.hasNext()) {
      ps.add(new Pair<Integer,Integer>(item,it2.next()));
    }
  }
  return ps;
}
```

Motiváló példa

```
package java.lang;
public interface CharSequence {
  int length();
  char charAt(int index);
  ...
}
```

Implementáló osztályok

- java.lang.String
- java.lang.StringBuilder
- java.lang.StringBuffer
- ...

Írjunk lexikografikus összehasonlítást!

```
static boolean less(CharSequence left, CharSequence right) {
```

Elegendő az altípusos polimorfizmus

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}
```

```
less("cool", "hot")

StringBuilder sb1 = new StringBuilder(); ...
StringBuilder sb2 = new StringBuilder(); ...
less(sb1, sb2)

less("cool", sb1)
```

ELTE IK

Nem elegendő az altípusos polimorfizmus

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}

static CharSequence min(CharSequence left,
                        CharSequence right) {
  return less(left,right) ? left : right;
}
```

Nem elegendő az altípusos polimorfizmus

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}

static CharSequence min(CharSequence left,
                        CharSequence right) {
  return less(left,right) ? left : right;
}
```

OK

```
CharSequence cs = min("cool", "hot");
```

Fordítási hiba

```
String str = min("cool", "hot");
```

Parametrikus polimorfizmus

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}

static <T> T min(T left, T right) {
  return less(left,right) ? left : right;
}
```

Fordítási hiba: `less`

Korlátozott univerzális kvantálás

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}
static <T extends CharSequence> T min(T left, T right) {
  return less(left,right) ? left : right;
}
```

ELTE
IK

Korlátozott univerzális kvantálás

```
static boolean less(CharSequence left, CharSequence right) {
  ...
}
static <T extends CharSequence> T min(T left, T right) {
  return less(left,right) ? left : right;
}
```

```
String str = min("cool", "hot");
StringBuilder sb = min(new StringBuilder(),
                       new StringBuilder());
CharSequence cs = min("cool", new StringBuilder());
```

ELTE
IK

Korlátozott univerzális kvantálás

- constrained genericity
- bounded universal quantification
- bounded parametric polymorphism
- $\forall T$-re, amely a `CharSequence`-ből származik, definiáljuk a `min` függvényt úgy, hogy...
- felső korlát (upper bound)

ELTE
IK

„Természetes rendezés” (natural ordering)

```
java.util.Arrays.sort(args)
```

„Természetes rendezés'' (natural ordering)

```
java.util.Arrays.sort(args)
```

```
public final class String ... {
  ...
  public int compareTo(String that) { ... }
}
```

```
public final class Integer ... {
  ...
  public int compareTo(Integer that) { ... }
}
```

IK

Természetes rendezés - interfész

Háromféle eredményű összehasonlítás

```
package java.lang;
public interface Comparable<T> {     // negatív: this < that
  int compareTo(T that);             //    nulla: this = that
}                                    // pozitív: this > that
```

Természetes rendezés - interfész

Háromféle eredményű összehasonlítás

```
package java.lang;
public interface Comparable<T> {     // negatív: this < that
  int compareTo(T that);             //    nulla: this = that
}                                    // pozitív: this > that
```

```
package java.lang;
public final class String implements Comparable<String> {...}
// public int compareTo(String that) { ... }
```

```
package java.lang;
public final class Integer implements Comparable<Integer>{...}
// public int compareTo(Integer that) { ... }
```

Saját természetes rendezés

Háromféle eredményű összehasonlítás

```
package java.lang;
public interface Comparable<T> {     // negatív: this < that
  int compareTo(T that);             //    nulla: this = that
}                                    // pozitív: this > that
```

```
public class Rational implements Comparable<Rational> {
  ...
  public int compareTo(Rational that) {
    // osztály invariánsa: denominator > 0
    return numerator * that.denominator -
           that.numerator * denominator;
  }
}
```

Rendezhetőség öröklése

```
public class Rational implements Comparable<Rational> {
  ...
  public int compareTo(Rational that) {
    // osztály invariánsa: denominator > 0
    return numerator * that.denominator -
           that.numerator * denominator;
  }
}
```

```
public class SimpleRational extends Rational { ... }
```

```
(new Rational(3,6)).compareTo(new Rational(5,9))
(new Rational(3,6)).compareTo(new SimpleRational(5,9))
(new SimpleRational(3,6)).compareTo(new SimpleRational(5,9))
(new SimpleRational(3,6)).compareTo(new SimpleRational(5,9))
```

Probléma az öröklődéssel

Egy osztály nem implementálhatja ugyanazt a generikus interfészt többször, különböző típusparaméterekkel.

```
public class Time implements Comparable<Time> {
  ...
  public int compareTo(Time that) { ... }
}
```

Fordítási hiba

```
public class ExactTime extends Time
                       implements Comparable<ExactTime> {
  ...
  public int compareTo(ExactTime that) { ... }
}
```

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
  int compare(T left, T right);   // 3-way
}
```

ELTE
IK

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
  int compare(T left, T right);   // 3-way
}
```

```
class StringLengthComparator implements Comparator<String> {
  public int compare(String left, String right) {
    return left.length() - right.length();
  }
}
```

```
java.util.Arrays.sort(args, new StringLengthComparator());
```

ELTE
IK

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
  int compare(T left, T right);   // 3-way
}
```

```
class StringLengthComparator implements Comparator<String> {
  public int compare(String left, String right) {
    return left.length() - right.length();
  }
}
```

```
java.util.Arrays.sort(args, new StringLengthComparator());
```

ELTE

```
java.util.Arrays.sort(args, (a,b) -> a.length()-b.length());
```

Rendezés

Nyilvános műveletek a `java.util.Arrays` osztályban:

```
static <T> void parallelSort(T[] a, Comparator<? super T> cmp)
```

- `cmp`: létezik olyan S típus, amelynek altípusa a T, és ilyeneket tud összehasonlítani
  - ◇ egzisztenciális kvantálás (existential quantification)
  - ◇ alsó korlát (lower bound)

ELTE
IK

Rendezés

Nyilvános műveletek a `java.util.Arrays` osztályban:

```
static <T> void parallelSort(T[] a, Comparator<? super T> cmp)
```

- `cmp`: létezik olyan S típus, amelynek altípusa a T, és ilyeneket tud összehasonlítani
  - ◇ egzisztenciális kvantálás (existential quantification)
  - ◇ alsó korlát (lower bound)

```
static <T extends Comparable<? super T>>
                                    void parallelSort(T[] a)
```

- A T olyan típus legyen, amelynek van olyan bázistípusa, amely rendelkezik természetes rendezéssel

ELTE
IK

Altípus reláció paraméterezett típusokon

```
public class ArrayList<T> ... implements List<T> ...
```

$\forall$ T-re: `ArrayList<T>` $<:$ `List<T>`

- `ArrayList<String>` $<:$ `List<String>`
- `ArrayList<Integer>` $<:$ `List<Integer>`

Típusparaméter altípusossága?

Szabály

`List<Integer>` $\not<:$ `List<Object>`

Típusparaméter altípusossága?

Szabály

`List<Integer>` $\not<:$ `List<Object>`

Indirekt tegyük fel, hogy `List<Integer>` $<:$ `List<Object>`

```
List<Integer> nums = new ArrayList<Integer>();
nums.add(42);                    // Integer.valueOf(42)
List<Object> things = nums;      // indirekt feltevés
things.add("forty-two");         // String <: Object
Integer n = nums.get(1);         // hiba!
```

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[]` $<:$ `Object[]` legyen!

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[] <: Object[]` legyen!

Paraméterezett típusokra

```
List<Integer> nums = new ArrayList<Integer>();
nums.add(42);                    // Integer.valueOf(42)
List<Object> things = nums;      // fordítási hiba
things.add("forty-two");         // String <: Object
Integer n = nums.get(1);         // hiba lenne
```

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[]` <: `Object[]` legyen!

Paraméterezett típusokra

```
List<Integer> nums = new ArrayList<Integer>();
nums.add(42);                    // Integer.valueOf(42)
List<Object> things = nums;      // fordítási hiba
things.add("forty-two");         // String <: Object
Integer n = nums.get(1);         // hiba lenne
```

Tömb típusokra

```
Integer[] nums = new Integer[2];
nums[0] =  42 ;                  // Integer.valueOf(42)
Object[] things = nums;          // szabályos
things[1] = "forty-two" ;        // ArrayStoreException
Integer n = nums[1];             // hiba lenne
```

Objektumok egyenlősége

```
Time t1 = new Time(13, 30);
Time t2 = new Time(13, 30);

System.out.println(t1 == t2);        // false
t2 = t1;
System.out.println(t1 == t2);        // true
                                     // memóriacím alapján
```

Tartalmi egyenlőségvizsgálat referenciatípusokra?

```
Time t1 = new Time(13, 30);
Time t2 = new Time(13, 30);

System.out.println(t1 == t2);       // false

System.out.println(t1.equals(t2)); // true
                                    // tartalom alapján
```

ELTE
IK

Tartalmi egyenlőségvizsgálat referenciatípusokra

```
ArrayList<Integer> seq1 = new ArrayList<>();
seq1.add(1984); seq1.add(2001);

ArrayList<Integer> seq2 = seq1;
System.out.println(seq1 == seq2);

seq2 = new ArrayList<>();
seq2.add(1984); seq2.add(2001);

System.out.println(seq1 == seq2);

System.out.println(seq1.equals(seq2));
```

ELTE
IK

„Az” equals metódus

```
package java.lang;
public class Object {
  ...
  public boolean equals(Object that) { ... }
}
```

- Felüldefiniálható (pl. `Time`-ra is)
- Egy metódus sok (rész)implementációval
- Együttesen adnak egy összetett implementációt

ELTE
IK

„Az” equals metódus

```
package java.lang;
public class Object {
  ...
  public boolean equals(Object that) { ... }
}
```

- Felüldefiniálható (pl. `Time`-ra is)
- Egy metódus sok (rész)implementációval
- Együttesen adnak egy összetett implementációt

- ... ha jól csináljuk!

Az `equals` szerződése betartandó

- Determinisztikus
- Ekvivalencia-reláció
  - ◇ RST: reflexív, szimmetrikus, tranzitív
- Ha `a != null`, akkor `!a.equals(null)`
  - ◇ Viszont `null.equals(a)` ⇒ `NullPointerException`
- Konzisztens a `hashCode()` metódussal
  - ◇ egyenlő objektumok `hashCode()`-ja egyezzen meg
  - ◇ [különböző objektumok `hashCode()`-ja jó, ha különböző]

Alapértelmezett viselkedés

```
package java.lang;
public class Object {
  ...

  public boolean equals(Object that) {
    return this == that;
  }

  public int hashCode() { ... }
}
```

Szabályos felüldefiniálás

```
public class Time {
  ...

  @Override public boolean equals(Object that) {

    if (that == null)                          return false;
    if (!getClass().equals(that.getClass()))   return false;
    Time t = (Time)that;
    return hour == t.hour && minute == t.minute;
  }

  @Override public int hashCode() { return 60*hour + minute; }
}
```

TK

Szabályos felüldefiniálás + „előző sáv”

```
public class Time {
  ...

  @Override public boolean equals(Object that) {
    if (this == that)                          return true;
    if (that == null)                          return false;
    if (!getClass().equals(that.getClass()))   return false;
    Time t = (Time)that;
    return hour == t.hour && minute == t.minute;
  }

  @Override public int hashCode() { return 60*hour + minute; }
}
```

TK

Jellemző hiba

```
package java.lang;
public class Object {
  ...
  public boolean equals(Object that) { return this == that; }
  public int hashCode() { ... }
}
```

Fordítási hiba a @Override-nak köszönhetően

```
public class Time {
  ...
  @Override public boolean equals(Time that) {
    return that != null && hour == that.hour && ...
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

Nagyon valószínű, hogy bug és egyben rossz gyakorlat

```
package java.lang;
public class Object {
  ...
  public boolean equals(Object that) { return this == that; }
  public int hashCode() { ... }
}
```

Túlterhelés (nincs dinamikus kötés)

```
public class Time {
  ...
  public boolean equals(Time that) {
    return that != null && hour == that.hour && ...
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

`@Override` jelentősége

- Explicit módon kifejezi a programozó szándékát
- A fordítóprogram szól, ha elrontottuk a felüldefiniálást

Használjuk!

ELTE IK

Túlterhelés altípuson

```
static void connect(Employee e, Manager m) {
  m.addUnderling(e);
}
static void connect(Manager m, Employee e) {
  m.addUnderling(e);
}
```

Túlterhelés altípuson

```
static void connect(Employee e, Manager m) {
  m.addUnderling(e);
}
static void connect(Manager m, Employee e) {
  m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);       connect(mary,eric);
```

Túlterhelés altípuson

```
static void connect(Employee e, Manager m) {
  m.addUnderling(e);
}
static void connect(Manager m, Employee e) {
  m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);       connect(mary,eric);

Manager mike = new Manager("Mike",13000);
connect(mike,mary);
```

ELTE
IK

Túlterhelés altípuson

```
static void connect(Employee e, Manager m) {
  m.addUnderling(e);
}
static void connect(Manager m, Employee e) {
  m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);       connect(mary,eric);

Manager mike = new Manager("Mike",13000);
connect(mike,mary);

connect(mike,(Employee)mary);
```

ELTE
IK

Ökölszabály

Soha ne terheljünk túl altípuson!

Ökölszabály

Soha ne terheljünk túl altípuson!

```
class Object {
  public boolean equals(Object that) { ... }
  ...
}
class Time {
  public boolean equals(Time that) { ... }
  ...
}
```

Ökölszabály

Soha ne terheljünk túl altípuson!

```
class Object {
  public boolean equals(Object that) { ... }
  ...
}
class Time {
  public boolean equals(Time that) { ... }
  ...
}

Time t = new Time(11,22);
Object o = new Time(11,22);

t.equals(t)     t.equals(o)     o.equals(t)     o.equals(o)
```

Öröklődés és equals

```
public class Time { ...
  @Override public boolean equals(Object that) {
    if (that == null)                              return false;
    if (!getClass().equals(that.getClass()))       return false;
    Time t = (Time)that;
    return hour == t.hour && minute == t.minute;
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

```
public class ExactTime extends Time { ...
  @Override public boolean equals(Object that) {
    return super.equals(that) && second == ((ExactTime)that).second
  }
  @Override public int hashCode() {
    return 60*super.hashCode() + second;
  }
```

Altípusosság?

```
public class Time {
  @Override public boolean equals(Object that) {
    ...
    return hour == t.hour && minute == t.minute;
  }
}
```

```
public class ExactTime extends Time {
  @Override public boolean equals(Object that) {
    return super.equals(that) && second == ((ExactTime)that).second
  }
}
```

```
new Time(11,22).equals(new ExactTime(11,22,33))
```

instanceof + „előző sáv”

```
public class Time {
  ...
  @Override public boolean equals(Object that) {
    if (this == that) return true;
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

ELTE
IK

instanceof

```
public class Time {
  ...
  @Override public boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

```
Time t = new Time(11,22);
ExactTime e = new ExactTime(11,22,33);

t.equals(e)
```

„Igazi” egyenlőségvizsgálat

```
public class Time {
  ...
  @Override public boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
  @Override public int hashCode() { return 60*hour + minute; }
}
```

```
ExactTime e1 = new ExactTime(11,22,44);
ExactTime e2 = new ExactTime(11,22,33);

e1.equals(e2)
```

Szimmetria?

```
public class Time {
  @Override public boolean equals(Object that) {
    if (that instanceof Time) { ...
      return hour == t.hour && minute == t.minute;
    ...
```

```
public class ExactTime extends Time {
  @Override public boolean equals(Object that) {
    if (that instanceof ExactTime) { ...
      ExactTime t = (ExactTime)that;
      return super.equals(that) && second == t.second;
    ...
```

Szimmetria?

```
public class Time {
  @Override public boolean equals(Object that) {
    if (that instanceof Time) { ...
      return hour == t.hour && minute == t.minute;
    ...
```

```
public class ExactTime extends Time {
  @Override public boolean equals(Object that) {
    if (that instanceof ExactTime) { ...
      ExactTime t = (ExactTime)that;
      return super.equals(that) && second == t.second;
    ...
```

```
Time t = new Time(11,22), e = new ExactTime(11,22,33);
t.equals(e)        e.equals(t)
```

Tranzitivitás?

```
public class ExactTime extends Time {
  @Override public boolean equals(Object that) {
    if (that instanceof ExactTime et) { ...
      return super.equals(that) && second == et.second;
    } else if (that instanceof Time) {
      return that.equals(this);
    } else {
      return false;
  ...
```

ELTE
IK

Tranzitivitás?

```
public class ExactTime extends Time {
  @Override public boolean equals(Object that) {
    if (that instanceof ExactTime et) { ...
      return super.equals(that) && second == et.second;
    } else if (that instanceof Time) {
      return that.equals(this);
    } else {
      return false;
  ...
```

```
Time t = new Time(11,22);
ExactTime e1 = new ExactTime(11,22,33);
ExactTime e2 = new ExactTime(11,22,44);
e1.equals(t)        t.equals(e2)        e1.equals(e2)
```

Öröklődés kiváltása kompozícióval

```
public class ExactTime {
  private final Time time;
  private int second;
  public ExactTime(int hour, int minute, int second) {
    time = new Time(hour,minute);
    if (0 <= second && second < 60) this.second = second;
    else throw new IllegalArgumentException();
  }
  public int getSecond() { return second; }
  public int getMinute() { return time.getMinute(); }
  public void aMinutePassed() { time.aMinutePassed(); }
  ...
}
```

ELTE
IK

Öröklődés kiváltása kompozícióval: egyenlőség

```
public final class ExactTime {
  private final Time time;
  private int second;
  ...
  @Override public boolean equals(Object that) {
    if (this == that) return true;
    if (that instanceof ExactTime et) {
      return time.equals(et.time) && second == et.second;
    }
    return false;
  }
}
```

ELTE
IK

final metódus

- nem definiálható felül

final metódus

- nem definiálható felül

```
public class Time {
  ...
  @Override public final boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
}
```

ELTE
IK

final metódus

- nem definiálható felül

```
public class Time {
  ...
  @Override public final boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
}
```

Fordítási hiba

```
public class ExactTime extends Time {
    @Override public boolean equals(Object that) ...
...
```

final metódus

- nem definiálható felül

```
public class Time {
  ...
  @Override public final boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
}
```

```
ExactTime e1 = new ExactTime(11,22,44);
ExactTime e2 = new ExactTime(11,22,33);
e1.equals(e2)  // RST, de nem ,,igazi" egyenlőség
```

final class

```
package java.lang;
public final class String implements ... { ... }
```

- Nem lehet belőle leszármaztatni
- Nem lehet specializálni, felüldefiniással belepiszkálni, elrontani
- Módosíthatatlan (immutable) esetben nagyon hasznos
- `java.lang.Class`, `java.lang.Integer` és egyéb csomagoló osztályok, `java.math.BigInteger` stb.

final class: végleges egyenlőségvizsgálat

```
public final class Time {
  ...
  @Override public boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
}
```

final class: végleges egyenlőségvizsgálat

```
public final class Time {
  ...
  @Override public boolean equals(Object that) {
    if (that instanceof Time t) {
      return hour == t.hour && minute == t.minute;
    }
    return false;
  }
}
```

Fordítási hiba

```
public class ExactTime extends Time { ... }
```

IK

Öröklődésre tervezés

- Ha azt akarjuk, hogy egy osztályból lehessen újabbakat származtatni, tervezzük olyanra!
  - ◇ `equals`
  - ◇ `protected` láthatóság
  - ◇ legyen jól dokumentált, hogyan kell származtatni belőle
  - ◇ legyen időtálló
- Ha nem akarjuk, hogy származtassanak belőle, tegyük `final`lé!

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun) // true
verb == noun      // true
```

ELTE
IK

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun) // true
verb == noun      // true

String mathematical = new String("ring");
noun.equals(mathematical) // true
noun == mathematical      // false
```

ELTE
IK

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun) // true
verb == noun      // true

String mathematical = new String("ring");
noun.equals(mathematical) // true
noun == mathematical      // false
```

Használjunk mindig `equals()`-t!

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne) // true
nineteen == twentyButOne      // true
```

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne) // true
nineteen == twentyButOne      // true

Integer dog = -123456;
Integer pup = -123456;

dog.equals(pup) // true
dog == pup      // false
```

ELTE
IK

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne) // true
nineteen == twentyButOne      // true

Integer dog = -123456;
Integer pup = -123456;

dog.equals(pup) // true
dog == pup      // false
```

Használjunk mindig `equals()`-t!

ELTE
IK

Felsorolási típusok egyenlőségvizsgálata

Használjunk mindig `equals()`-t!

ELTE
IK

Felsorolási típusok egyenlőségvizsgálata

Használjunk mindig `equals()`-t!

- ... vannak esetek, amikor nem muszáj

ELTE IK

Felsorolási típusok egyenlőségvizsgálata

Használjunk mindig `equals()`-t!

- ... vannak esetek, amikor nem muszáj

Felsorolási típus

- Garantáltan működik az `==` is.

```
enum Color { RED, WHITE, GREEN }
...
if (color1 == color2) ...
```

ELTE IK

Felsorolási típusok egyenlőségvizsgálata

Használjunk mindig `equals()`-t!

- ... vannak esetek, amikor nem muszáj

Felsorolási típus

- Garantáltan működik az `==` is.

```
enum Color { RED, WHITE, GREEN }
...
if (color1 == color2) ...
```

- Nem példányosítható
- Nem származtatható le belőle
- Használható `switch`-utasításban

ELTE IK

Heterogén egyenlőség

```
ArrayList<Integer> aList = new ArrayList<>();
LinkedList<Integer> lList = new LinkedList<>();

aList.add(19);
lList.add(20-1);

aList.equals(lList)
```

ELTE
IK

Adatszerkezetek

- `ArrayList`, `HashSet`, `HashMap`
- Az `equals` és a `hashCode` helyességén alapszanak

```
ArrayList.contains(item)
HashSet.add(item)
HashMap.get(key)
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []
```

ELTE
IK

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []

set.add(t);
t.setHour(6);
set.remove(new Time(5,30));
System.out.println(set);                          // [6:30]
```

ELTE
IK

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []

set.add(t);
t.setHour(6);
set.remove(new Time(5,30));
System.out.println(set);                          // [6:30]

set.remove(new Time(6,30));
System.out.println(set);                          // [6:30]
```

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)
```

ELTE
IK

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(int[] x, int[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (x[i] != y[i]) { return false; }
  }
  return true;
}
```

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)
```

```
static boolean isEqualTo(int[] x, int[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (x[i] != y[i]) { return false; }
  }
  return true;
}
```

```
java.util.Arrays.equals(x,y)
```

Referenciák tömbjének összehasonlítása

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)
```

Referenciák tömbjének összehasonlítása

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(Integer[] x, Integer[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (!x[i].equals(y[i]))
      return false;
  }
  return true;
}
```

ELTE
IK

Referenciák tömbjének összehasonlítása: pontosabban

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(Integer[] x, Integer[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (x[i] != y[i] && (x[i] == null || !x[i].equals(y[i])))
      return false;
  }
  return true;
}
```

ELTE
IK

Referenciák tömbjének összehasonlítása: pontosabban

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(Integer[] x, Integer[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (x[i] != y[i] && (x[i] == null || !x[i].equals(y[i])))
      return false;
  }
  return true;
}
```

```
java.util.Arrays.equals(x,y)
```

Referenciák tömbjének összehasonlítása: pontosabban, v2

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(Integer[] x, Integer[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (Objects.equals(x[i], y[i]))
      return false;
  }
  return true;
}
```

Referenciák tömbjének összehasonlítása: pontosabban, v2

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean isEqualTo(Integer[] x, Integer[] y) {
  if (x == y) { return true; }
  if (x == null || y == null || x.length != y.length) {
    return false;
  }
  for (int i=0; i<x.length; ++i) {
    if (Objects.equals(x[i], y[i]))
      return false;
  }
  return true;
}
```

```
java.util.Arrays.equals(x,y)
```

Tömbök tömbjének összehasonlítása

```
int[][] x = {{1,2}}, y = {{1,2}};
! x.equals(y)
! java.util.Arrays.equals(x,y)
java.util.Arrays.deepEquals(x,y)
```

ELTE
IK

Tömbök tömbjének összehasonlítása

```
int[][] x = {{1,2}}, y = {{1,2}};
! x.equals(y)
! java.util.Arrays.equals(x,y)
java.util.Arrays.deepEquals(x,y)

int[][][] x = {{{1,2}}}, y = {{{1,2}}};
java.util.Arrays.deepEquals(x,y)
```

ELTE
IK

Egyenlőségvizsgálat primitív mezők esetén

```
public class Time {
  ...

  @Override public boolean equals(Object that) {
    if (that == null)                          return false;
    if (!getClass().equals(that.getClass()))   return false;

    Time t = (Time)that;
    return hour == t.hour && minute == t.minute;
  }

  @Override public int hashCode() { return 60*hour + minute; }
}
```

Mély vizsgálat referencia típusú mezőkre

```
public class Interval {
  private Time from;
  private Time to;
  ...
  @Override public boolean equals(Object that) {
    if (that == null)                          return false;
    if (!getClass().equals(that.getClass()))   return false;

    Interval u = (Interval)that;
    return from.equals(u.from) && to.equals(u.to);
  }
  ...
}
```

- Csak ha `from` és `to` nem lehet `null`!

ELTE
IK

java.util.Objects osztály

```
java.util.Objects.equals(Object a, Object b)
java.util.Objects.deepEquals(Object a, Object b)
java.util.Objects.hash(Object a...)
```

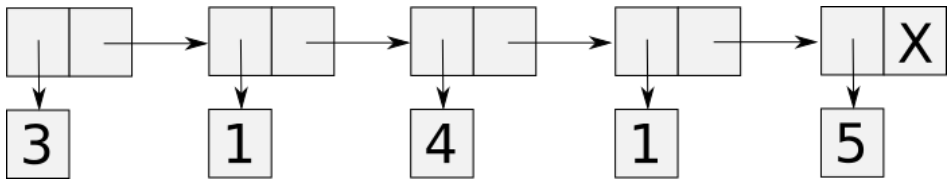
null-okat toleráló equals és jó hashCode

```
import java.util.Objects;
public class Interval {
  private Time from, to;   // nulls are allowed
  ...
  @Override public boolean equals(Object that) {
    if (that == null)                          return false;
    if (!getClass().equals(that.getClass()))   return false;

    Interval u = (Interval)that;
    return Objects.equals(from, u.from) &&
           Objects.equals(to, u.to);
  }
  @Override public int hashCode() {
    return Objects.hash(from, to);
  }
}
```

Saját fejlesztésű *sorozat* osztály

```
package datastructures;
public class Sequence<E> {
    public void insert(int index, E element) { ... }
    public E get(int index) { ... }
    public E remove(int index) { ... }
    public int length() { ... }
}
```



Megvalósítás láncolt listával

datastructures/Sequence.java

```
package datastructures;
public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...
}
```

datastructures/Node.java

```
package datastructures;
class Node<E> {
    E data;
    Node<E> next;
    Node(E data, Node<E> next) { ... }
}
```

Egy fordítási egységben több típusdefiníció

- Még mindig szükségtelenül sokan hozzáférnek a segédosztályhoz

datastructures/Sequence.java

```
package datastructures;
public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...
}

class Node<E> {
    E data;
    Node<E> next;
    Node(E data, Node<E> next) { ... }
}
```

Privát statikus tagosztály

datastructures/Sequence.java

```
package datastructures;

public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...

    private static class Node<E> {
        E data;
        Node<E> next;
        Node(E data, Node<E> next) { ... }
    }
}
```

Statikus típusbeágyazás: java.util.Map.Entry

```
package java.util;

public interface Map<K,V> {

        public static interface Entry<K,V> {
                ...
        }

        ...

}
```

ELTE
IK

Bejáró statikus tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() { return new SeqIt<>(this);

    private static class SeqIt<E> implements Iterator<E> {
        private Node<E> current;
        SeqIt(Sequence<E> seq) { current = seq.first; }
        public boolean hasNext() { return current != null; }
        public E next() { ... }
    }
}
```

ELTE
IK

Példányszintű beágyazás

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() { return new SeqIt<>(this);

    private static class SeqIt<E> implements Iterator<E> {
        private Node<E> current;
        SeqIt(Sequence<E> seq) { current = seq.first; }
        public boolean hasNext() { return current != null; }
        public E next() { ... }
    }
}
```

ELTE
IK

Bejáró példányszintű tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() { return new SeqIt(); }

    private class SeqIt implements Iterator<E> {
        private Node<E> current = first;
        public boolean hasNext() { return current != null; }
        public E next() { ... }
    }
}
```

ELTE
IK

Bejáró példányszintű tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() { return new SeqIt(); }

    private class SeqIt implements Iterator<E> {
        private Node<E> current = first;
        public boolean hasNext() { return current != null; }
        public E next() { ... }
    }
}
```

- Sequence.this.first

Bejáró lokális osztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() {
        class SeqIt implements Iterator<E> {
            private Node<E> current = first;
            public boolean hasNext() { return current != null;
            public E next() { ... }
        };
        return new SeqIt();
    }
}
```

Bejáró névtelen osztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator() {
        return new Iterator<E>() {
            private Node<E> current = first;
            public boolean hasNext() { return current != null;
            public E next() { ... }
        };
    }
}
```

Lambdák

```
@FunctionalInterface
public interface Comparator<T> {
  int compare(T left, T right);
}
```

```
java.util.Arrays.sort(args, (a,b) -> a.length()-b.length());
```

Lambdák

```
@FunctionalInterface
public interface Comparator<T> {
  int compare(T left, T right);
}
```

```
java.util.Arrays.sort(args, (a,b) -> a.length()-b.length());
```

```
java.util.Arrays.sort(
  args,
  new Comparator<String>() {
    public int compare(String left, String right) {
      return left.length() - right.length();
    }
  }
);
```

Bájtkód

- `datastructures.Sequence.Node`: `datastructures/Sequence$Node.class`
- `datastructures.Sequence` első névtelen osztálya: `datastructures/Sequence$1.class`

Lambda-kifejezések

```
int[] nats = {0, 1, 2, 3, 4, 5, 6};
```

```
int[] nats = new int[1000];
for (int i=0; i<nats.length; ++i) nats[i] = i;
```

```
int[] nats = new int[1000];
java.util.Arrays.setAll(nats, i->i);

java.util.Arrays.setAll(nats, i->(int)(100*Math.random()));
```

Több paraméterrel

```
public static void main(String[] args) {
    java.util.Arrays.sort(args);
    java.util.Arrays.sort(args,
        (s,z) -> s.length()-z.length());
}
```

ELTE
IK

Lehetőségek

```
i -> i

(int i) -> i+1

(int n, String s) -> {  StringBuilder sb = new StringBuilder();
                        for (int i=1; i<=n; ++i) sb.append(s);
                        return sb.toString();
                     }
```

Funkcionális programozás: ízelítő

```
int[] nums = new int[1000];
java.util.Arrays.setAll(nums, i->(int)(100*Math.random()));

java.util.Arrays.stream(nums)
                .filter(i -> i%2 == 0)
                .map(i -> i/2)
                .limit(10)
                .forEach(i -> System.out.println(i))
```

Részleges alkalmazás

```
java.util.Arrays.stream(nums)
                .forEach(i -> System.out.println(i))
```

```
java.util.Arrays.stream(nums)
                .forEach(System.out::println)
```

Interfész lambdák típusozásához

```
interface IntIntToInt {
  int apply(int left, int right);
}
```

```
static int[] zipWith(IntIntToInt fun, int[] left, int[] right)
  int[] result = new int[Integer.min(left.length, right.length
  for (int i=0; i<result.length; ++i) {
    result[i] = fun.apply(left[i], right[i]);
  }
  return result;
}
```

```
zipWith( (n, m) -> n*m, new int[]{1,2,3}, new int[]{6,5,4} )
```