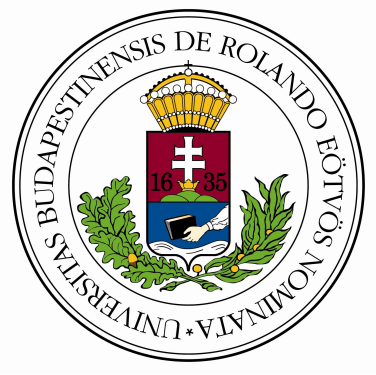
Programozási nyelvek – Java

Egyenlőségvizsgálat



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Object identity

```
Time t1 = new Time(13,30);
Time t2 = new Time(13,30);
System.out.println( t1 == t2 );

t2 = t1;
System.out.println( t1 == t2 );
```

Tartalmi egyenlőségvizsgálat referenciatípusokra?

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

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

System.out.println( t1.equals(t2) );
```

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) );
```

„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

„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)
- Ha `a != null`, akkor `!a.equals(null)`
  - Viszont `null.equals(a)` $\equiv$ `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 && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }

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

Szabályos felüldefiniálás + „előző sáv”

```
public class Time {
    ...

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

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

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!

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);
```

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);
```

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

```
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 ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

instanceof

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

Tranzitivitás?

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

`final` metódus

- nem definiálható felül

final metódus

- nem definiálható felül

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

final metódus

- nem definiálható felül

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

final class: végleges egyenlőségvizsgálat

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

Fordítási hiba

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

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!

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(); }
    ...
}
```

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 ){
            ExactTime et = (ExactTime) that;
            return time.equals(et.time) && second == et.second;
        } else return false;
    }
}
```

Heterogén egyenlőség

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

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

aList.equals(lList)
```

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);  // []
```

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]
```

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]
```

Stringek egyenlőségvizsgálata

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

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

Stringek egyenlőségvizsgálata

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

verb.equals(noun)
verb == noun

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

Stringek egyenlőségvizsgálata

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

verb.equals(noun)
verb == noun

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

Használjunk mindig equals()-t!

Integerek egyenlőségvizsgálata

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

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

Integerek egyenlőségvizsgálata

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

nineteen.equals(twentyButOne)
nineteen == twentyButOne

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

dog.equals(pup)
dog == pup
```

Integerek egyenlőségvizsgálata

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

nineteen.equals(twentyButOne)
nineteen == twentyButOne

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

dog.equals(pup)
dog == pup
```

Használjunk mindig equals()-t!

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Felsorolási típus

- Garantáltan működik az == is.

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

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Felsorolási típus

- Garantáltan működik az == is.

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

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

Tömbök összehasonlítása

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

Tömbök összehasonlítása

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

static boolean is_equal( 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 is_equal( 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 is_equal( 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;
}
```

Referenciák tömbjének összehasonlítása: pontosabban

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

static boolean is_equal( 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;
}
```

Referenciák tömbjének összehasonlítása: pontosabban

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

```
static boolean is_equal( 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)
```

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)
```

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)
```

Egyenlőségvizsgálat primitív mezők esetén

```
public class Time {
    ...

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

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

Mély vizsgálat referencia típusú mezőkre

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

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

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