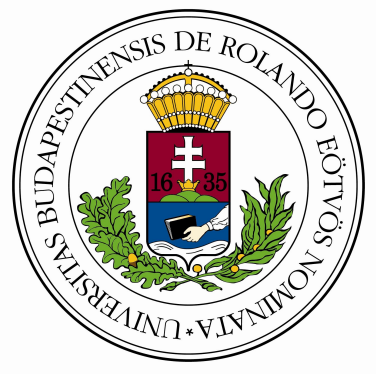
Programozási nyelvek – Java

Negyedik előadás



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Racionális számok

```
package numbers;
public class Rational {

    private int numerator, denominator;
    /* class invariant: denominator > 0 */

    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

}
```

Getter-setter

```
package numbers;
public class Rational {

    private int numerator, denominator;

    public Rational( int numerator, int denominator ){ ... }

    public void setDenominator( int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.denominator = denominator;
    }

    public int getDenominator(){ return denominator; }

    ...
}
```

Aritmetika

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public void setNumerator( int numerator ){ ... }
    public void setDenominator( int denominator ){ ... }

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
    ...
}
```

Dokumentációs megjegyzéssel

```
package numbers;
public class Rational {
    ...
    /**
    *  Set {@code this} to {@code this} * {@code that}.
    *  @param that Non-null reference to a rational number,
    *              it will not be changed in the method.
    *  @throws NullPointerException When {@code that} is null.
    */
    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
    ...
}
```

Főprogram

```
import numbers.Rational.*;
public class Main {
    public static void main( String[] args ){
        Rational p = new Rational(1,3);
        Rational q = new Rational(1,2);
        p.multiplyWith(q);
        println(p);
        println(q);
    }
    private static void println( Rational r ){
        System.out.println( r.getNumerator() + "/" +
                            r.getDenominator() );
    }
}
```

Műveletek sorozása

```
package numbers;
public class Rational {
    ...
    public Rational multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
        return this;
    }
    ...
}
```

```
Rational p = new Rational(1,3);
Rational q = new Rational(1,2);
p.multiplyWith(q).multiplyWith(q).divideBy(q);
println(p);
```

Teljesen másfajta megoldás

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    ...

    public Rational times( Rational that ){
        return new Rational( this.numerator * that.numerator,
                             this.denominator * that.denominator );
    }

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
}
```

Használjuk mindkettőt

```
package numbers;
public class Rational {
    ...
    public void multiplyWith( Rational that ){ ... }
    public Rational times( Rational that ){ ... }
}
```

```
Rational p = new Rational(1,3);
Rational q = new Rational(1,2);
p.multiplyWith(q).multiplyWith(q).divideBy(q);
println(p);
Rational r = p.times(q);
println(r);
println(p);
```

Funkcionális stílus

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational  plus( Rational that ){ ... }
    ...
}
```

Módosíthatatlan mezőkkel

```
package numbers;
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational  plus( Rational that ){ ... }
    ...
}
```

Módosíthatatlan változó

```
final int width = 80;
```

- Ha egyszer értéket kapott, nem adhatunk új értéket neki
- A deklarációban értéket kell kapjon
- Hasonló a C-beli const-hoz (de nem pont ugyanaz)

lokális változó, formális paraméter

Globális konstans

```
public static final int WIDTH = 80;
```

- Osztályszintű mező
- Picit olyan, mint a C-ben egy #define
- Konvenció: végig nagy betűvel írjuk a nevét

Módosíthatatlan mező

- Például `WIDTH` globális konstans
- Vagy `Rational` két mezője
- Ha egyszer értéket kapott, nem adhatunk új értéket neki
- Inicializáció során értéket kell kapjon
  - „Üres konstans” (blank final)!

Mutable versus Immutable

Módosítható belső állapot (OOP)

```
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }     ...
    public void setNumerator( int numerator ){ ... }   ...
    public void multiplyWith( Rational that ){
```

Módosíthatatlan belső állapot (FP)

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
```

Nyilvános módosíthatatlan belső állapot

```
public class Rational {
    public final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public Rational times( Rational that ){ ... }
    ...
}
```

Érzékeny a reprezentációváltoztatásra!

Más elnevezési konvenció

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int numerator(){ return numerator; }
    public int denominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
}
```

```
System.out.println( p.numerator() + "/" + p.denominator() );
```

Reprezentációváltás

```
public class Rational {
    private final int[] data;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        data = new int[]{ numerator, denominator };
    }
    public int numerator(){ return data[0]; }
    public int denominator(){ return data[1]; }
    public Rational times( Rational that ){ ... }
}
```

final referencia

```
final int[] data = new int[2];
data[0] = 3;
data[0] = 4;
data = new int[3]; // fordítási hiba
```

Karaktersorozatok ábrázolása

- java.lang.String: módosíthatatlan (immutable)

```
String fourtytwo = "42";
String twentyfour = fourtytwo.reverse();
String twentyfourhundredfourtytwo = twentyfour + fourtytwo;
```

- java.lang.StringBuider és java.lang.StringBuffer: módosítható

```
StringBuilder sb = new StringBuilder("");
for( char c = 'a'; c <= 'z'; ++c ){
   sb.append(c).append(',');
}
sb.deleteCharAt(sb.length()-1);    // remove last comma
String letters = sb.toString();
```

- char[]: módosítható

Hatékonyságbeli kérdés

```
StringBuilder sb = new StringBuilder("");    // temporary
for( char c = 'a'; c <= 'z'; ++c ){
   sb.append(c).append(',');
}
sb.deleteCharAt(sb.length()-1);
String letters = sb.toString();
```

```
String letters = "";
for( char c = 'a'; c <= 'z'; ++c ){
   letters += (c + ',');
}
letters = letters.substring(0,letters.length()-1);
```

Procedurális stílus (függvény)

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int numerator(){ return numerator; }
    public int denominator(){ return denominator; }

    public static Rational times( Rational left, Rational right ){
        return new Rational( left.numerator * right.numerator,
                             left.denominator * right.denominator );
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational r = Rational.times(p,q);
```

Procedurális stílus (eljárás)

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public void setNumerator( int numerator ){ ... }
    ...
    public static void multiplyLeftWithRight( Rational left,
                                              Rational right ){
        left.numerator *= right.numerator;
        left.denominator *= right.denominator;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational.multiplyLeftWithRight(p,q);
```

Paraméterátadás Javában

Érték szerinti (call-by-name)

- primitív típusú paraméterre

```
public void setNumerator( int numerator ){
    this.numerator = numerator;
}
```

Megosztás szerinti (call-by-sharing)

- referencia típusú paraméterre
- a referenciát érték szerint adjuk át

```
public static void multiplyLeftWithRight( Rational left,
                                          Rational right ){
    left.numerator *= right.numerator;
    left.denominator *= right.denominator;
}
```

Érték szerinti (call-by-name)

```
public void setNumerator( int numerator ){
    this.numerator = numerator;
    numerator = 0;
}
```

```
Rational p = new Rational(1,3);
int two = 2;
p.setNumerator(two);
println(p);
System.out.println(two);
```

Megosztás szerinti (call-by-sharing)

```
public static void multiplyLeftWithRight( Rational left,
                                          Rational right ){
    left.numerator *= right.numerator;
    left.denominator *= right.denominator;
    left = new Rational(9,7);
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational.multiplyLeftWithRight(p,q);
println(p);
```

Ha a paraméterek nem diszjunktak...

```
package numbers;
public class Rational {
    ...

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }

    public void divideBy( Rational that ){
        if( that.numerator == 0 )
            throw new ArithmeticException("Division by zero!");
        this.numerator *= that.denominator;
        this.denominator *= that.numerator;
    }
}
```

Belső állapot kiszivárgása

```
public class Rational {
    private final int[] data;
    ...
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public void set( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = data;
    }
}
```

```
Rational p = new Rational(1,2);
int[] cheat = {3,4};
p.set(cheat);
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Belső állapot kiszivárgása ügyetlen konstruálás miatt

```
public class Rational {
    private final int[] data;
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public Rational( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = data;
    }
}
```

```
int[] cheat = {3,4};
Rational p = new Rational(p);
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Belső állapot kiszivárgása getteren keresztül

```
public class Rational {
    private final int[] data;
    ...
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public int[] get(){ return data; }
}
```

```
Rational p = new Rational(1,2);
int[] cheat = p.get();
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Defenzív másolás

```
public class Rational {
    private final int[] data;
    public Rational( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = new int[]{ data[0], data[1] };
    }
    public void set( int[] data ){ /* hasonlóan */ }
    public int[] get(){
        return new int[]{ data[0], data[1] };
    }
}
```

Módosíthatatlan objektumokat nem kell másolni

```
public class Person {
    private String name;
    private int age;
    public Person( String name, int age ){
        if( name == null || name.trim().isEmpty() || age < 0 )
            throw new IllegalArgumentException();
        this.data = name;
        this.age = age;
    }
    public String getName(){ return name; }
    public int getAge(){ return age; }
    public void setName( String name ){ ... this.name = name; }
    public void setAge( int age ){ ... this.age = age; }
}
```

Tömbelemek között is lehet aliasing

```
Rational rats[2]; // fordítási hiba

Rational rats[] = new Rational[2]; // = {null,null};

Rational[] rats = new Rational[2]; // gyakoribb
rats[0] = new Rational(1,2);
rats[1] = rats[0];
rats[1].setDenominator(3);
System.out.println(rats[0].getDenominator());
```

- módosítható versus módosíthatatlan

Ugyanaz az objektum többször is lehet a tömbben

```
/**
    ...
    PRE: rats != null
    ...
*/
public static void increaseAllByOne( Rational[] rats ){
    for( Rational p: rats ){
        p.setNumerator( p.getNumerator() + p.getDenominator() );
    }
}
```

Dokumentálva

```
/**
    ...
    PRE: rats != null and (i!=j => rats[i] != rats[j])
    ...
*/
public static void increaseAllByOne( Rational[] rats ){
    for( Rational p: rats ){
        p.setNumerator( p.getNumerator() + p.getDenominator() );
    }
}
```

Tömbök tömbje

- Javában nincs többdimenziós tömb (sor- vagy oszlopfolytonos)
- Tömbök tömbje (referenciák tömbje)

```
int[][] matrix = {{1,0,0},{0,1,0},{0,0,1}};

int[][] matrix = new int[3][3];
for( int i=0; i<matrix.length; ++i ) matrix[i][i] = 1;

int[][] matrix = new int[5][];
for( int i=0; i<matrix.length; ++i ) matrix[i] = new int[i];
```

Ismét aliasing – bug-gyanús

```
Rational[][] matrix = { {new Rational(1,2), new Rational(1,2)},
                        {new Rational(1,2), new Rational(1,2)},
                        {new Rational(1,2), new Rational(1,2)} };
```

```
Rational half = new Rational(1,2);
Rational[] halves = {half, half};
Rational[][] matrix = {halves, halves, halves};
```

Több metódus ugyanazzal a névvel

```
public class Rational {
    ...

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }

    public void multiplyWith( int that ){
        this.numerator *= that.numerator;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
p.multiplyWith(q);
p.multiplyWith(2);
```

Több konstruktor ugyanabban az osztályban

```
public class Rational {
    ...

    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public Rational( int value ){
        numerator = value;
        denominator = 1;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(3);
```

Túlterhelés

- Több metódus ugyanazzal a névvel, több konstruktor
- Formális paraméterek eltérnek
  - Paraméterek száma
  - Paraméterek deklarált típusa
- Híváskor a fordító eldönti, melyiket kell hívni
  - Az aktuális paraméterek száma,
  - illetve deklarált típusa alapján
- Fordítási hiba, ha:
  - Egyik sem felel meg a hívásnak
  - Ha több is egyformán megfelel

Konstruktorok egymást hívhatják

```
public class Rational {
    ...
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public Rational( int value ){
        this(value,1);
    }

    public Rational(){
        this(0);
    }
}
```

Alapértelmezett érték

```
public class Rational {
    ...
    public void set( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public void set( int value ){
        set(value,1);
    }

    public void set(){
        set(0);
    }
}
```

Outline

Egy korábbi példa

```
public class Receptionist {
    ...
    public Time[] readWakeupTimes( String[] fnames ){
        Time[] times = new Time[fnames.length];
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times[i] = readTime(fnames[i]);
            } catch( java.io.IOException e ){
                times[i] = null;   // no-op
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // maybe sort times before returning?
    }
}
```

A null értékek kiszűrése

```
public class Receptionist {
    ...
    public Time[] readWakeupTimes( String[] fnames ){
        Time[] times = new Time[fnames.length];
        int j = 0;
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times[j] = readTime(fnames[i]);
                ++j;
            } catch( java.io.IOException e ){
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return java.util.Arrays.copyOf(times,j); // possibly sort
    }
}
```

Tömbök előnyei és hátrányai

- Elemek hatékony elérése (indexelés)
- Szintaktikus támogatás a nyelvben (indexelés, tömbliterál)
- Fix hossz: létrehozáskor
  - Bővítéshez új tömb létrehozása + másolás
  - Törléshez új tömb létrehozása + másolás

Alternatíva: java.util.ArrayList

kényelmes szabványos könyvtár, hasonló belső működés

```
String[] names = { "Tim",
                   "Jerry" };
```

```
ArrayList<String> names =
            new ArrayList<>();
names.add("Tim");
names.add("Jerry");
```

```
names[0] = "Tom";
String mouse = names[1];
```

```
names.set(0, "Tom");
String mouse = names.get(1);
```

```
String trio = new String[3];
trio[0] = names[0];
trio[1] = names[1];
trio[2] = "Spike";
names = trio;
```

```
names.add("Spike");
```

Az előző példa átalakítva

```
public class Receptionist {
    ...
    public ArrayList<Time> readWakeupTimes( String[] fnames ){
        ArrayList<Time> times = new ArrayList<Time>();
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times.add( readTime(fnames[i]) );
            } catch( java.io.IOException e ){
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // possibly sort before returning
    }
}
```

Paraméterezett típus

```
ArrayList<Time> times
```

```
Time[] times
Time times[]
```

Paraméterezés típussal

```
length :: [a] -> Int
length (x:xs) = 1 + length xs
length [] = 0

length [1..10] + length ["alma", "a", "fa", "alatt"]


reverse :: [a] -> [a]
reverse (x:xs) = reverse xs ++ [x]
reverse [] = []
```

Generikus osztály

Nem pont így, de hasonlóan...!

```
package java.util;
public class ArrayList<T> {
    public ArrayList(){ ... }
    public T get( int index ){ ... }
    public void set( int index, T item ){ ... }
    public void add( T item ){ ... }
    ...
}
```

Használatkor típusparaméter megadása

```
import java.util.ArrayList;

...

ArrayList<Time> times;
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> names = new ArrayList<>();
```

Generikus metódus

```
import java.util.*;
class Main {
    public static <T> void reverse( T[] array ){
        int lo = 0, hi = array.length-1;
        while( lo < hi ){
            T tmp = array[hi];
            array[hi] = array[lo];
            array[lo] = tmp;
            ++lo; --hi;
        }
    }
    public static void main( String[] args ){
        reverse(args);
        System.out.println( Arrays.toString(args) );
    }
}
```

Parametrikus polimorfizmus

- Több típusra is működik ugyanaz a kód
  - Java: típus (osztály), metódus
- Típussal paraméterezhető kód
  - Java: referenciatípusokkal

Típusparaméter

Helytelen

```
ArrayList<int> numbers
```

Helyes

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add( Integer.valueOf(7) );
Integer seven = numbers.get(0);

numbers.add(42);
int fourtytwo = numbers.get(1);
```

Outline

Típuskonverziók primitív típusok között

Automatikus típuskonverzió

- `byte` < `short` < `int` < `long`
- `long` < `float`
- `float` < `double`
- `char` < `int`
- `byte b = 42;` és `short s = 42;` és `char c = 42;`

Explicit típuskényszerítés (type cast)

```
int i = 42;
short s = (short)i;
```

Puzzle 3: Long Division (Bloch & Gafter: Java Puzzlers)

```
public class LongDivision {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

Csomagoló osztályok

Implicit importált (`java.lang`), immutable osztályok

- `java.lang.Boolean` – `boolean`
- `java.lang.Character` – `char`
- `java.lang.Byte` – `byte`
- `java.lang.Short` – `short`
- `java.lang.Integer` – `int`
- `java.lang.Long` – `long`
- `java.lang.Float` – `float`
- `java.lang.Double` – `double`

java.lang.Integer interfésze (részlet)

```
static int MAX_VALUE    // 2^31-1
static int MIN_VALUE    // -2^31

static int compare( int x, int y )    // 3-way comparison
static int max( int x, int y )
static int min( int x, int y )
static int parseInt( String str [, int radix] )
static String toString( int i [, int radix] )
static Integer valueOf( int i )

int compareTo( Integer that )        // 3-way comparison
int intValue()
```

Auto-(un)boxing

- Automatikus kétirányú konverzió
- Primitív típus és a csomagoló osztálya között

```
Integer ref = 42;
int pri = ref;

Integer sum = ref + pri;
```

```
Integer ref = Integer.valueOf(42);
int pri = ref.intValue();

Integer sum = Integer.valueOf (
                ref.intValue()
                + pri           );
```

Auto-(un)boxing + generikusok

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(7);
int seven = numbers.get(0);
```

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add( Integer.valueOf(7) );
int seven = numbers.get(0).intValue();
```

Számolás egész számokkal

```
int n = 10;
int fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Rosszul használt auto-(un)boxing

```
Integer n = 10;
Integer fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Jelentés

```
Integer n = Integer.valueOf(10);
Integer fact = Integer.valueOf(1);
while( n.intValue() > 1 ){
    fact = Integer.valueOf(fact.intValue() * n.intValue());
    n = Integer.valueOf(n.intValue() - 1);
}
```