Programozási nyelvek – Java

Ötödik előadás

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

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 (tranzitív)

- `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);
}
```

Outline

Absztrakció: egységbe zárás és információ elrejtése

```
public class Rational {
    private final int numerator, denominator;
    private static int gcd( int a, int b ){ ... }
    private void simplify(){ ... }
    public Rational( int numerator, int denominator ){ ... }
    public Rational( int value ){ super(value,1); }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational times( int that ){ ... }
    public Rational plus( Rational that ){ ... }
    ...
}
```

Egy osztály interfésze

```
public Rational( int numerator, int denominator )
public Rational( int value )
public int getNumerator()
public int getDenominator()
public Rational times( Rational that )
public Rational times( int that )
public Rational plus( Rational that )
...
```

Az interface-definíció

```
public interface Rational {
    public int getNumerator();
    public int getDenominator();
    public Rational times( Rational that );
    public Rational times( int that );
    public Rational plus( Rational that );
    ...
}
```

interface: automatikusan publikusak a tagok

```
public interface Rational {
    int getNumerator();
    int getDenominator();
    Rational times( Rational that );
    Rational times( int that );
    Rational plus( Rational that );
    ...
}
```

Az interface-definíció tartalma

Példánymetódusok deklarációja: specifikáció és ;

```
int getNumerator();
```

Az `interface`-definíció tartalma, de tényleg

- Példánymetódusok deklarációja
  - Esetleg `default` implementáció
- Konstansok definíciója: `public static final`
- Statikus metódus
- Beágyazott (tag-) típus

Interface megvalósítása

Rational.java

```
public interface Rational {
    int getNumerator();
    int getDenominator();
    Rational times( Rational that );
}
```

Fraction.java

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

Több megvalósítás

Simplified.java

```
public class Simplified implements Rational {
    ...
    public int getNumerator(){ ... }
    public int getDenominator(){ ... }
    Rational times( Rational that ){ ... }
}
```

Fraction.java

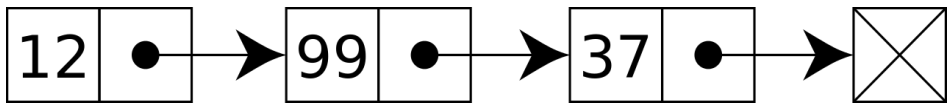
```
public class Fraction implements Rational {
    private final int numerator, denominator;
    public Fraction( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that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

Sorozat típusok ismét

- `int[]`
- `java.util.ArrayList<Integer>`
- `java.util.LinkedList<Integer>`

Láncolt ábrázolás

```
public class LinkedList<T> {
    private T head;
    private LinkedList<T> tail;
    public LinkedList(){ ... }
    public T get( int index ){ ... }
    public void set( int index, T item ){ ... }
    public void add( T item ){ ... }
    ...
}
```



Generikus interface

java/util/List.java

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

java/util/ArrayList.java

```
package java.util;
public class ArrayList<T> implements List<T>{
    public ArrayList(){ ... }
    public T get( int index ){ ... }
    ...
}
```

Altípusosság

```
class Fraction implements Rational { ... }
class ArrayList<T> implements List<T> { ... }
```

- `Fraction` <: `Rational`
- `Simplified` <: `Rational`
- Minden `T`-re: `ArrayList<T>` <: `List<T>`
- Minden `T`-re: `LinkedList<T>` <: `List<T>`

Liskov-féle helyettesítési elv

LSP: Liskov's Substitution Principle

Egy A típus altípusa a B (bázis-)típusnak, ha az A egyedeit használhatjuk a B egyedei helyett, anélkül, hogy ebből baj lenne.

Az interface egy típus

```
List<String> names;

static List<String> noDups( List<String> names ){
    ...
}
```

Nem példányosítható

```
List<String> names = new List<String>();    // fordítási hiba
```

Az osztály is egy típus, és példányosítható

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> nicks = new ArrayList<>();
```

Típusozás interface-szel, példányosítás osztállyal

```
List<String> names = new ArrayList<>();
```

Jó stílus...

Statikus és dinamikus típus

Változó (vagy paraméter) „deklarált”, illetve „tényleges” típusa

```
List<String> names = new ArrayList<>();


static List<String> noDups( List<String> names ){
    ... names ...
}


List<String> shortList = noDups(names);
```

Speciális jelentésű interface-ek

```
class DataStructure<T> implements java.lang.Iterable<T>
// működik rá az iteráló 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
```