

Programozási nyelvek – Java

Interface



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

1 Törzsanyag

2 Szorgalmi anyag

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 );  
    ...  
}
```



abstract műveletek: csak deklaráljuk őket

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

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 ){ ... }  
}
```

Simplified.java

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

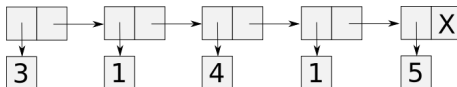
Sorozat típusok ismét

- `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> { ... }  
class LinkedList<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
```



Iterable és Iterator

- Iterálható (pl. egy adatszerkezet): ha kérhetünk tőle iterátort
- Iterátor: az adatszerkezet elemeinek egymás utáni lekérdezéséhez

```
java.lang.Iterable
```

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

```
java.util.Iterator
```

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

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);
```



Iteráló 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);
```



Többszörös iterálás

```
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;  
}
```



Interface 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} )
```



1 Törzsanyag

2 Szorgalmi anyag

Functional Interface és @FunctionalInterface

- Lambdák típusa lehet
- Csak egy implementálandó metódus
- Speciális *annotáció* használható

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



További példák

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

```
java.util.Arrays.setAll
```

```
public static void setAll( int[] array, IntUnaryOperator op )
```

```
package java.util.function;
```

```
@FunctionalInterface public interface IntUnaryOperator {  
    int applyAsInt( int operand );  
    ...  
}
```



Streamek

```
java.util.Arrays.stream
```

```
public static IntStream stream( int[] array )
```

```
package java.util.stream;
```

```
public interface IntStream {  
    IntStream filter( IntPredicate predicate );  
    IntStream map( IntUnaryOperator mapper );  
    int reduce( int identity, IntBinaryOperator op );  
    void forEach( IntConsumer action );  
    int[] toArray();  
    ...  
}
```



Mi lehet egy interface-ben?

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



Globális konstansok interface-ekben

```
package java.awt;  
  
public interface Transparency {  
  
    public static final    int OPAQUE = 1;  
    /*public static final*/ int BITMASK = 2;  
    /*public static final*/ int TRANSLUCENT = 3;  
  
    int getTransparency();  
  
}
```



Globális konstans versus enum

```
interface Color {  
    int RED = 0, WHITE = 1, GREEN = 2; // public static final  
    ...  
}
```

```
enum Color { RED, WHITE, GREEN }
```



default és static metódusok, tagtípus

```
package java.util.stream;

public interface IntStream {

    ...
    default IntStream dropWhile( IntPredicate predicate ){

        ...

    }
    static IntStream iterate( int seed, IntUnaryOperator f ){

        ...

    }
    public static interface Builder {
        void accept( int v );

        ...

    }
}
```



Marker interface

- Metódusok nélküli
- Mégis jelent valamit

```
package java.lang;  
public interface Cloneable {}
```

```
package java.io;  
public interface Serializable {}
```



Annotációtípusok

`@Documented`

`@Retention(RUNTIME)`

`@Target(TYPE)`

`public @interface FunctionalInterface {}`

