

# Programozási nyelvek – Java

Harmadik előadás



**Kozsik Tamás**

ELTE Eötvös Loránd Tudományegyetem

- Objektum-elvű programozás
  - Osztály és objektum
  - Egységbe zárás
  - Információ elrejtése
- Memóriakezelés
  - Referenciák
  - Végrehajtási verem és dinamikus tárhely
  - Példányszintű és osztályszintű tagok
  - Inicializáció
- Csomagok
- Fordítás és futtatás



## 1 Java programok szerkezete

## 2 Hibák és kivételek

- Kivételkezelés

## 3 Metódusok, konstruktorok

- Variációk egy osztályra
- `final` változók
- Osztályszintű metódusok
- Paraméterátadás
- Túlterhelés

# Forráskód felépítése

- fordítási egységek
- típusdefiníciók
- metódusok
- utasítások
- kifejezések
- lexikális elemek
- karakterek



# Karakterek

## Karakterkódolási szabványok (character encodings)

- Bacon's cipher, 1605 (Francis Bacon)
- Baude-code, 1874
- BCDIC, 1928 (Binary Coded Decimal Interchange Code)
- EBCDIC, 1963 (Extended ...)
- ASCII, 1963 (American Standard Code for Information Interchange)
- ISO/IEC 8859 (Latin-1, Latin-2,...)
- Windows 1250 (Cp1250)
- Unicode (UTF-8, UTF-16, UTF-32)

lásd: `iconv` (Unix/Linux)



# Lexikális elemek

- Kulcsszavak (while, case, class, new stb.)
- Azonosítók (pl. Point, move)
- Operátorok (<=, =, <<< stb.)
- Literálok (pl. 6.022140857E23, "hello", '\n')
- Zárójelek, speciális jelek
- Megjegyzések (egysoros, többsoros, „dokumentációs”)



# Kifejezések

- szintaxis: operátorok arítása, fixitása; zárójelezés
- kiértékelés
  - precedencia ( $A + B * C$ )
  - asszociativitás ( $A - B - C$ )
  - operandusok kiértékelési sorrendje ( $A + B$ )
  - lustaság ( $A \& B$  és  $A \&\& B$ )
  - mellékhatás ( $++x$ )



# Utasítások

- Értékadások
- Metódushívás
- return-utasítás
- Elágazások (if, switch)
- Ciklusok (while, do-while, for)
- Blokk-utasítás
- Változódeklaráció
- Kivételkezelő és -kiváltó utasítások
- assert-utasítás





# Metódusok

- Végrehajtási verem, aktivációs rekord
- Paraméterátadás [!]
- Osztályszintű és példányszintű
- Hatókör (lokális változók), elfedés
- Láthatósági kategóriák
- Inicializáció: konstruktor
- Túlterhelés [!]
- Példánymetódusok felüldefiniálása [!!]



# Típusdefiníciók

- **Osztály** (class)
- Interfész (interface)
- Felsorolási típus (enum)
- Annotáció típus (@interface)

(egymásba ágyazás)



# Fordítási egység

compilation unit

- opcionális package utasítás
- opcionális import utasítások
- típusdefiníciók



# Az import utasítás

- Teljes név helyett rövid név
- Más, mint az #include a C-ben!

## Típusnév importálása

```
import java.io.FileReader;  
...  
FileReader f;
```

## Minden típusnév egy csomagból

```
import java.io.*;  
...  
FileReader in;  
FileWriter out;
```



# Statikus tagok importálása

```
import static java.util.Arrays.sort;
class Main {
    public static void main( String[] args ){
        sort(args);
        for( String s: args ){
            System.out.println(s);
        }
    }
}
```



## 1 Java programok szerkezete

## 2 Hibák és kivételek

- Kivételkezelés

## 3 Metódusok, konstruktorok

- Variációk egy osztályra
- `final` változók
- Osztályszintű metódusok
- Paraméterátadás
- Túlterhelés

# Hiba detektálása és jelzése

```
public class Time {  
    private int hour;           // 0 <= hour < 24  
    private int minute;        // 0 <= minute < 60  
    public Time( int hour, int minute ){ ... }  
    public int getHour(){ return hour; }  
    public int getMinute(){ return minute; }  
    public void setHour( int hour ){  
        if( 0 <= hour && hour <= 23 ){  
            this.hour = hour;  
        } else {  
            throw new IllegalArgumentException("Invalid hour!");  
        }  
    }  
    public void setMinute( int minute ){ ... }  
    public void oneMinutePassed(){ ... }  
}
```



# Az assert utasítás

```
public class Time {  
    private int hour;           // 0 <= hour < 24  
    private int minute;        // 0 <= minute < 60  
    public Time( int hour, int minute ){ ... }  
    public int getHour(){ return hour; }  
    public int getMinute(){ return minute; }  
  
    // may throw AssertionError  
    public void setHour( int hour ){  
        assert 0 <= hour && hour <= 23 ;  
        this.hour = hour;  
    }  
  
    public void setMinute( int minute ){ ... }  
    public void oneMinutePassed(){ ... }  
}
```





# Az assert utasítás

## TestTime.java

```
Time time = new Time(6,30);  
time.setHour(30);
```

## Futtatás

```
$ java TestTime  
$ java -enableassertions TestTime  
Exception in thread "main" java.lang.AssertionError  
    at Time.setHour(Time.java:7)  
    at TestTime.main(TestTime.java:5)  
$
```



# Dokumentációs megjegyzés

```
/** May throw AssertionError. */  
public void setHour( int hour ){  
    assert 0 <= hour && hour <= 23 ;  
    this.hour = hour;  
}
```



# Dokumentált potenciálisan hibás használat

```
/**
```

Blindly sets the hour property to the given value.  
Use it with care: only pass {@code hour} satisfying  
{@code 0 <= hour && hour <= 23}.

```
*/
```

```
public void setHour( int hour ){  
    this.hour = hour;  
}
```



# javadoc Time.java

PACKAGE **CLASS** TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

## Constructor Summary

### Constructors

Constructor	Description
<code>Time()</code>	

## Method Summary

### All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
int	<code>getHour()</code>	
int	<code>getMinute()</code>	
void	<code>oneMinutePassed()</code>	
void	<code>setHour(int hour)</code>	Blindly sets the hour property to the given value.



## javadoc Time.java

PACKAGE **CLASS** TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SEARCH: 

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

**getHour**

```
public int getHour()
```

**getMinute**

```
public int getMinute()
```

**setHour**

```
public void setHour(int hour)
```

Blindly sets the hour property to the given value. Use it with care: only pass hour satisfying  $0 \leq \text{hour} \leq 23$ .



# Szokásos (túl bőbeszédű) dokumentációs megjegyzés

```
/**
 * Sets the hour property. Only pass an {@code hour}
 * satisfying {@code 0 <= hour && hour <= 23}.
 * @param hour The value to be set.
 * @throws IllegalArgumentException
 *     If the supplied value is not between 0 and 23,
 *     inclusively.
 */
public void setHour( int hour ){
    if( 0 <= hour && hour <= 23 ){
        this.hour = hour;
    } else {
        throw new IllegalArgumentException("Invalid hour!");
    }
}
```



# javadoc Time.java

## setHour

```
public void setHour(int hour)
```

Sets the hour property. Only pass an hour satisfying  $0 \leq \text{hour} \leq 23$ .

### Parameters:

hour - The value to be set.

### Throws:

`java.lang.IllegalArgumentException` - If the supplied value is not between 0 and 23, inclusively.



# Szintaxiskiemelés

```
/**
 * Sets the hour property. Only pass an {@code hour}
 * satisfying {@code 0 <= hour && hour <= 23}.
 * @param hour The value to be set.
 * @throws IllegalArgumentException
 *     If the supplied value is not between 0 and 23,
 *     inclusively.
 */
public void setHour( int hour ){
    if( 0 <= hour && hour <= 23 ){
        this.hour = hour;
    } else {
        throw new IllegalArgumentException("Invalid hour!");
    }
}
```

21,1





# Opciók hibák jelzésére

## Jó megoldások

- `IllegalArgumentException`: modul határán
- `assert`: modul belsejében
- Dokumentációs megjegyzés

## Rossz megoldások

- Csendben elszabotálni a műveletet
- Elsumákolni az ellenőrzéseket



# Ellenőrzött kivételek

checked exceptions

```
public Time readTime( String fname ) throws java.io.IOException {  
    ...  
}
```

- A programszövegben jelölni kell a terjedését
- A fordítóprogram ellenőrzi a konzisztenciát
- Ilyen: `java.sql.SQLException`, `java.security.KeyException`
- Nem ilyen: `NullPointerException`, `ArrayIndexOutOfBoundsException`



# Terjedés követése: fordítási hiba

```
import java.io.IOException;
class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ){
        TestTime tt = new TestTime();
        Time wakeUp = tt.readTime("wakeup.txt");
        wakeUp.oneMinutePassed();
    }
}
```



# Terjedés követése: fordítási hiba javítva

```
import java.io.IOException;
class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ) throws IOException {
        TestTime tt = new TestTime();
        Time wakeUp = tt.readTime("wakeup.txt");
        wakeUp.oneMinutePassed();
    }
}
```



# Kivételkezelés

```
import java.io.IOException;

class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ){
        TestTime tt = new TestTime();
        try {
            Time wakeUp = tt.readTime("wakeUp.txt");
            wakeUp.oneMinutePassed();
        } catch( IOException e ){
            System.err.println("Could not read wake-up time.");
        }
    }
}
```



# A program tovább futhat a probléma ellenére

```
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 try-catch utasítás

```
<try-catch-statement> ::= try <block-statement>  
                           <catch-list>  
                           <optional-finally-part>
```

```
<catch-list> ::= <catch-part>  
                | <catch-part> <catch-list>
```

```
<catch-part> ::= catch (<exceptions> <identifier>)  
                    <block-statement>
```

```
<exceptions> ::= <identifier>  
                | <identifier> | <exceptions>
```

```
<optional-finally-part> ::= ""  
                          | finally <block-statement>
```



# Több catch-ág

```
public static Time parse( String str ){
    String errorMessage;
    try {
        String[] parts = str.split(":");
        int hour = Integer.parseInt(parts[0]);
        int minute = Integer.parseInt(parts[1]);
        return new Time(hour,minute);
    } catch( NullPointerException e ){
        errorMessage = "Null parameter is not allowed!";
    } catch( ArrayIndexOutOfBoundsException e ){
        errorMessage = "String must contain \":\"!";
    } catch( NumberFormatException e ){
        errorMessage = "String must contain two numbers!";
    }
    throw new IllegalArgumentException(errorMessage);
}
```





# Egy catch-ágban több kivétel

```
public static Time parse( String str ){  
    try {  
        String[] parts = str.split(":");  
        int hour = Integer.parseInt(parts[0]);  
        int minute = Integer.parseInt(parts[1]);  
        return new Time(hour,minute);  
    } catch( NullPointerException  
            | ArrayIndexOutOfBoundsException  
            | NumberFormatException e ){  
        throw new IllegalArgumentException("Can't parse time!");  
    }  
}
```



# A try-finally utasítás

```
public static Time readTime( String fname ) throws IOException {  
    BufferedReader in = new BufferedReader(new FileReader(fname));  
    Time time;  
    try {  
        String line = in.readLine();  
        time = parse(line);  
    } finally {  
        in.close();  
    }  
    return time;  
}
```



# A finally mindenképp vezérlést kap!

```
public static Time readTime( String fname ) throws IOException {  
    BufferedReader in = new BufferedReader(new FileReader(fname));  
    try {  
        String line = in.readLine();  
        return parse(line);  
    } finally {  
        in.close();  
    }  
}
```



# A try-catch-finally utasítás

```
public static Time readTime( String fname ) throws IOException {  
    BufferedReader in = new BufferedReader(new FileReader(fname));  
    try {  
        String line = in.readLine();  
        return parse(line);  
    } catch ( IllegalArgumentException e ){  
        System.err.println(e);  
        System.err.println("Using default value!");  
        return new Time(0,0);  
    } finally {  
        in.close();  
    }  
}
```



# A try-utasítások egymásba ágyazhatók

```
public static Time readTimeOrUseDefault( String fname ){  
    try {  
        BufferedReader in =  
            new BufferedReader(new FileReader(fname));  
        try {  
            String line = in.readLine();  
            return parse(line);  
        } finally {  
            in.close();  
        }  
    } catch( IOException | IllegalArgumentException e ){  
        System.err.println(e);  
        System.err.println("Using default value!");  
        return new Time(0,0);  
    }  
}
```



# A *try-with-resources* utasítás

```
public static Time readTimeOrUseDefault( String fname ){  
    try {  
        try(  
            BufferedReader in =  
                new BufferedReader(new FileReader(fname))  
        ){  
            String line = in.readLine();  
            return parse(line);  
        }  
    } catch( IOException | IllegalArgumentException e ){  
        System.err.println(e);  
        System.err.println("Using default value!");  
        return new Time(0,0);  
    }  
}
```



# Lényegében ekvivalensek

## try-finally

```
BufferedReader in = ... ;  
try {  
    String line = in.readLine();  
    return parse(line);  
} finally {  
    in.close();  
}
```

## try-with-resources

```
try(  
    BufferedReader in = ...  
) {  
    String line = in.readLine();  
    return parse(line);  
}
```



# Bonyolultabb eset: fájl másolása

```
static void copy( String in, String out ) throws IOException {  
    try (  
        FileInputStream infile = new FileInputStream(in);  
        FileOutputStream outfile = new FileOutputStream(out)  
    ){  
        int b;  
        while( (b = infile.read()) != -1 ){    // idióma!  
            outfile.write(b);  
        }  
    }  
}
```





## 1 Java programok szerkezete

## 2 Hibák és kivételek

- Kivételkezelés

## 3 Metódusok, konstruktorok

- Variációk egy osztályra
- `final` változók
- Osztályszintű metódusok
- Paraméterátadás
- Túlterhelés

# 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() );
```



# 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);
```



# Aliasing probléma!

```
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;
    }
}
```



# 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);  
    }  
}
```

