Programozási nyelvek Java

Kódszervezés

Kozsik Tamás

Hibajelzés kivétel kiváltásával

```
public class Time {
  int hour;                              // 0 <= hour < 24
  int min;                               // 0 <= min  < 60

  ...

  public void setHour(int hour) {
    if (0 <= hour && hour <= 23) {
      this.hour = hour;
    } else {
      throw new IllegalArgumentException("Invalid hour!");
    }
  }
}
```

Az assert utasítás

```
public class Time {
  int hour;                               // 0 <= hour < 24
  int min;                                // 0 <= min  < 60

  ...

  public void setHour(int hour) {
    assert 0 <= hour && hour <= 23;
    this.hour = hour;
  }
```

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)
$
```

ELTE
IK

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 exception

```
public Time readTime(String fname) throws java.io.IOException
    // ez a kódrészlet kiválthat IOException kivételt
}
```

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

ELTE
IK

Ellenőrzött kivételek

Checked exception

```
public Time readTime(String fname) throws java.io.IOException
    // ez a kódrészlet kiválthat IOException kivételt
}
```

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

Unchecked exception

- Pl. `NullPointerException`, `ArrayIndexOutOfBoundsException`
- Dinamikus szemantikai hiba
- „Bárhol” keletkezhet

Terjedés követése: fordítási hiba

```
import java.io.IOException;
public 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.aMinutePassed();
  }
}
```

Terjedés követése: fordítási hiba javítva

```
import java.io.IOException;
public 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.aMinutePassed();
  }
}
```

Kivételkezelés

```
import java.io.IOException;
public 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.aMinutePassed();
    } 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?
  }
}
```

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

```java
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
  var 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
  var 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 {
  var 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();
  }
}
```

ELTE
IK

A try-utasítások egymásba ágyazhatók

```
public static Time readTimeOrUseDefault(String fn) {
  try {
    var in = new BufferedReader(new FileReader(fn));
    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);
  }
}
```

Erőforráskezelő try (*try-with-resources*) utasítás

```java
public static Time readTimeOrUseDefault(String fn) {
  try (
    var in = new BufferedReader(new FileReader(fn))
  ) {
    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);
}
```

Több erőforrás használata

```
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);
    }
  }
}
```

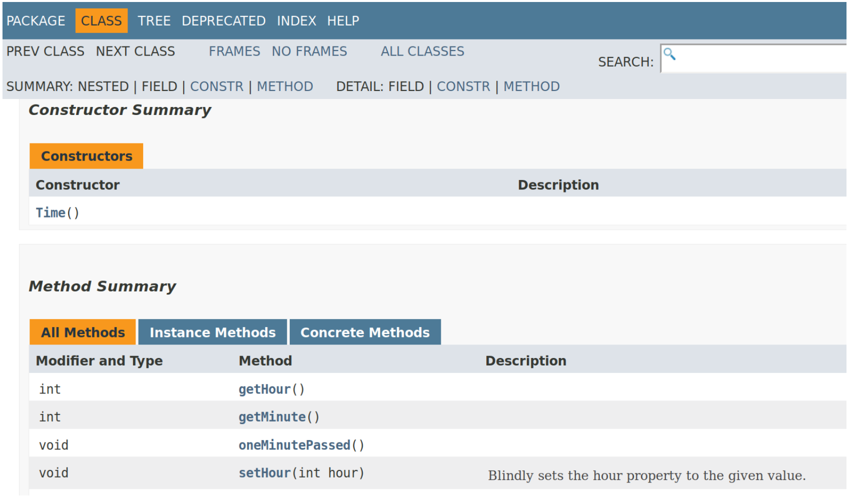
Dokumentációs megjegyzés

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

Dokumentált potenciálisan hibás használat

```java
/**
  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

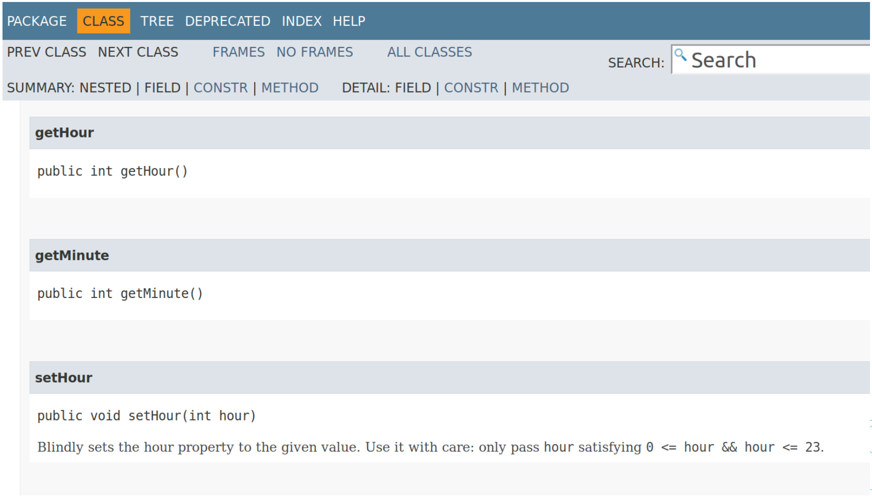
Constructor	Description
`Time()`	

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method	Description
`int`	`getHour()`	
`int`	`getMinute()`	
`void`	`oneMinutePassed()`	
`void`	`setHour(int hour)`	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: Search
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

getHour

```
public int getHour()
```

getMinute

```
public int getMinute()
```

setHour

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

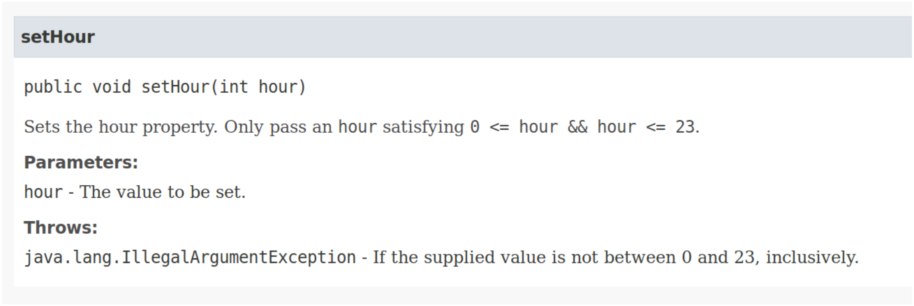
Blindly sets the hour property to the given value. Use it with care: only pass `hour` satisfying `0 <= hour && hour <= 23`.

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!");
    }
}
```

ELTE
IK

javadoc Time.java



setHour

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

Sets the hour property. Only pass an hour satisfying `0 <= hour && hour <= 23`.

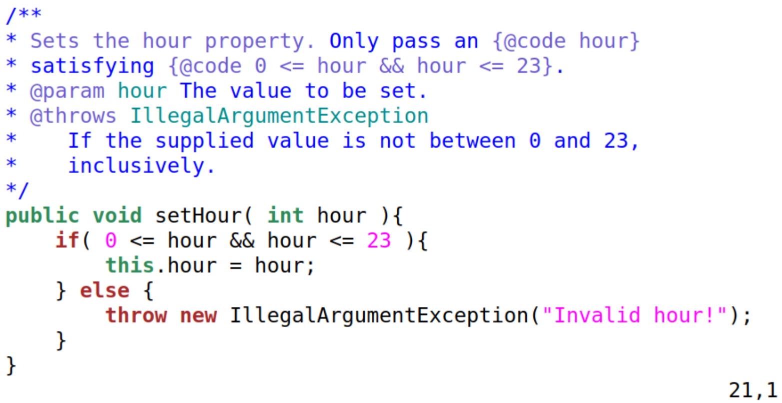
Parameters:

`hour` - The value to be set.

Throws:

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

Syntax highlighting



```
/**
* 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
```

Racionális számok megvalósítása

```
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; }

  ...
}
```

Tervezett használat

```
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);               // 1/6
    println(q);               // 1/2
  }
  private static void println(Rational r) {
      System.out.println(r.getNumerator()+"/"+r.getDenominator());
  }
}
```

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;
  }
  ...
}
```

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);
```

Osztályszintű metódus (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);
```

Osztályszintű metódus (eljárás)

```
public class Rational {
  private int numerator, denominator;
  ...
  public static void multiplyInPlace(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);
```

Egy másfajta megközelítés

```
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);
println(p);             // 1/6
Rational r = p.times(q);
println(r);             // 1/12
println(p);             // 1/6
```

Megvalósítások

```
package numbers;
public class Rational {
  private int numerator;
  private int 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;
  }
}
```

Megvalósítások

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

Operátor-túlterhelés nincs a Javában

```
package numbers;
public class Rational {
  private int numerator;
  private int denominator;
  public Rational(int numerator, int denominator) { ... }
  ...
  public Rational operator*(Rational that) { // compilation error
      return new Rational(this.numerator * that.numerator,
                          this.denominator * that.denominator);
  }
  public Rational operator*=(Rational that) { // compilation error
    this.numerator *= that.numerator;
    this.denominator *= that.denominator;
    return this;
  }
}
```

Sosem módosuló belső állapot

```java
package numbers;
public class Rational {
  private int numerator;
  private int 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) { ... }
  ...
}
```

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;
  }
}
```

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

Trükkös szabályok: „jobban illeszkedő''

```
static void m(long n) { ... }
static void m(float n) { ... }
public static void main(String[] args) {
  m(3);
}
```

Egyformán illeszkedő

```
static void m(long n, float m) { ... }
static void m(float m, long n) { ... }
public static void main(String[] args) {
  m(4,2);
}
```

```
Foo.java:5: error: reference to m is ambiguous
        m(4,2);
        ^
  both method m(long,float) in Foo
   and method m(float,long) in Foo match
1 error
```

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 (overloading)

- Több metódus ugyanazzal a névvel, több konstruktor

Túlterhelés (overloading)

- Több metódus ugyanazzal a névvel, több konstruktor
- Formális paraméterek eltérnek
  - ◇ Paraméterek száma
  - ◇ Paraméterek deklarált típusa

ELTE
IK

Túlterhelés (overloading)

- 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

Túlterhelés (overloading)

- 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
  - ◇ Több is egyformán megfelel

Túlterhelés (overloading)

- 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
  - ◇ Több is egyformán megfelel
- Angolul nem összekeverendő: over**rid**ing (felüldefiniálás) vs over**load**ing (túlterhelés)

Jó ez így?

```
public class Rational {
  ...

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

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

Jogos túlterhelés

```
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(Rational that) {
      if (that == null) throw new IllegalArgumentException();
      this.numerator = that.numerator;
      this.denominator = that.denominator;
  }
  ...
}
```

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);
  }
  ...
}
```

Alapértelmezett érték – a Java ezt nem engedi

```
public class Rational {
  ...
  public Rational(int numerator = 0, int denominator = 1) {
    if (denominator <= 0) throw new IllegalArgumentException();
    this.numerator = numerator;
    this.denominator = denominator;
  }

  public void set(int numerator = 0, int denominator = 1) {
    if (denominator <= 0) throw new IllegalArgumentException();
    this.numerator = numerator;
    this.denominator = denominator;
  }
  ...
}
```

Konstruktorok hívhatják egymást

```
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); // a legelső utasításnak kell lennie!
  }

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

Konstruktor(ok) helyett gyártóművelet(ek)

e.g. `Rational.zero()` instead of `new Rational(0)`

```
public class Rational {
  ...
  private Rational(int numerator, int denominator) {
    this.numerator = numerator;
    this.denominator = denominator;
  }
  public static Rational make(int numerator, int denominator) {
    return new Rational(numerator,denominator);
  }
  public static Rational valueOf(int val) {return make(val, 1);}
  public static Rational oneOver(int den) {return make(1, den);}
  public static Rational zero() { return make(0, 1); }
}
```

Paraméterátadási technikák

- Szövegszerű helyettesítés
- Érték szerinti
- Érték-eredmény szerinti
- Eredmény szerinti
- Cím szerinti
- Megosztás szerinti
- Név szerinti
- Igény szerinti

Paraméterátadás Javában

Érték szerinti (call-by-value)

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-value)

```
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)

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

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

Változó számú paraméter

```
static int sum(int[] nums) {
  int sum = 0;
  for (int num: nums) { sum += num; }
  return sum;
}
                         sum(new int[]{1,2,3,4,5,6})
```

Változó számú paraméter

```
static int sum(int[] nums) {
  int sum = 0;
  for (int num: nums) { sum += num; }
  return sum;
}
                         sum(new int[]{1,2,3,4,5,6})
```

```
static int sum(int... nums) {
    int sum = 0;
    for (int num: nums) { sum += num; }
    return sum;
}
                         sum(new int[]{1,2,3,4,5,6})
```

Globális konstans

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

- Osztályszintű mező
- Picit olyan, mint a C-ben egy `#define`
- Hasonló a C-beli `const`-hoz is (de nem pont ugyanaz)
- Konvenció: végig nagybetűvel 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)!

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

Módosíthatatlan lokális változó

```
public class Rational {
  ...
  public void simplify() {
    final int gcd = gcd(numerator, denominator);
    numerator /= gcd;
    denominator /= gcd;
  }
  ...
}
```

Módosíthatatlan formális paraméter

Hibás

```
static java.math.BigInteger factorial(final int n) {
  assert n > 0;
  java.math.BigInteger result = java.math.BigInteger.ONE;
  while (n > 1) {
    result = result.multiply(java.math.BigInteger.valueOf(n))
    --n;
  }
  return result;
}
```

Módosíthatatlan formális paraméter

Helyes

```
static java.math.BigInteger factorial(final int n) {
  assert n > 0;
  java.math.BigInteger result = java.math.BigInteger.ONE;
  for (int i=n; i>1; --i) {
    result = result.multiply(java.math.BigInteger.valueOf(i))

  }
  return result;
}
```

Mutable versus Immutable

Módosítható belső állapot

```
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

```
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!

Reprezentációváltás

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

Kitérő

```
int[] t = new int[3];
t = new int[4];

int[] s = {1,2,3};
s = {1,2,3,4};  // compilation error
s = new int[]{1,2,3,4};
```

final hivatkozás

```
final Rational p = new Rational(1,2);
p.setNumerator(3);
p = new Rational(1,4); // compilation error
```

final hivatkozás

```
final Rational p = new Rational(1,2);
p.setNumerator(3);
p = new Rational(1,4); // compilation error

final int[] data = new int[2];
data[0] = 3;
data[1] = 4;
data = new int[3]; // compilation error
```

ELTE
IK

Karaktersorozatok ábrázolása

- java.lang.String: módosíthatatlan (immutable)

```
String num42 = "42";
String num24 = num42.reverse();
String num4224 = num42 + num24;
```

Karaktersorozatok ábrázolása

- `java.lang.String`: módosíthatatlan (immutable)

```
String num42 = "42";
String num24 = num42.reverse();
String num4224 = num42 + num24;
```

- `java.lang.StringBuilder` (és `StringBuffer`): módosítható

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

Karaktersorozatok ábrázolása

- java.lang.String: módosíthatatlan (immutable)

```
String num42 = "42";
String num24 = num42.reverse();
String num4224 = num42 + num24;
```

- java.lang.StringBuilder (és StringBuffer): módosítható

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

- char[]: módosítható

Hatékonyságbeli kérdés

```
StringBuilder sb = new StringBuilder("");
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);
```

Íme egy jól kinéző osztálydefiníció...

```
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;
  }
}
```

És ha a paraméterek nem diszjunktak?

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

```
Rational p = new Rational(1,2);
p.divideBy(p);
```

Belső állapot kiszivárgása

```
public class Rational {
  private 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;
  }
}
```

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

Belső állapot kiszivárgása *getter*en 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) { /* similarly */ }
  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.name = 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 r: rats) {
    r.setNumerator(r.getNumerator() + r.getDenominator());
  }
}
```

Dokumentálva

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