

Programozási nyelvek II.: JAVA

2. gyakorlat

2017. szeptember 18-22.

Emlékeztető: Hello World!

```
/**
 * Fordítás: javac HelloWorld.java
 * Futtatás: java HelloWorld
 */

public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello World!");
        //kiiras a standard outputra
    }
}
```

A 2. gyakorlat tematikája

- Osztályok és objektumok
- Csomagok, importálás, saját csomag készítése
- Fordítás, futtatás
- Programok nyomonkövetése

- Osztály (típus) = séma:
 - objektumok reprezentációjának megadása
- Objektum: egy osztály egy példánya
 - minden objektum valamilyen osztályból származik példányosítással
- Reprezentáció:
 - példányadattagok, példány metódusok

A komplex számok osztálya (ComplexPolar.java)

- Készítsük el a komplex számok polárkoordinátás (trigonometrikus) alakjának reprezentációját!
- Tulajdonságok: modulus és szögérték (emlékeztető: a $z = a + bi$ komplex szám felírható $z = r \cdot (\cos \varphi + i \cdot \sin \varphi)$ alakban, ahol r nemnegatív szám z modulusa, a φ pedig a szögértéke (radiánban). Ekkor $a = r \cos \varphi$ és $b = r \sin \varphi$.
- Műveletek: (1) egy komplex szám konjugáltjának, (2) két komplex szám összegének, különbségének és szorzatának, (3) egy komplex szám n -dik hatványának (ahol n nemnegatív szám) kiszámítása, valamint (4) egy komplex szám String-ként való ábrázolásának megadása. (A szorzat: $z_1 \cdot z_2 = r_1 r_2 (\cos(\varphi + \psi) + i \sin(\varphi + \psi))$), az n -dik hatvány: $z^n = r^n (\cos(n\varphi) + i \sin(n\varphi))$.)

A komplex számok osztálya (ComplexPolar.java)

```
/**
 * Komplex számok polarkoordinatas (trigonometrikus) alakja.
 */

public class ComplexPolar {
    public double r;           // nemnegativ szam
                              // a komplex szam modulusa
    public double theta;     // szogertek

    //a komplex szam valos reszenek kiszamitasa
    public double re() { return r * Math.cos(theta); }
    //a komplex szam kepzetes reszenek kiszamitasa
    public double im() { return r * Math.sin(theta); }
    ...
}
```

A komplex számok osztálya (ComplexPolar.java)

```
...
// komplex számok algebrai alakját
// polarkoordinatassa alakítja
public ComplexPolar toPolar (double re, double im) {
    ComplexPolar a = new ComplexPolar();
    a.r      = Math.sqrt(re*re + im*im);
    a.theta = Math.atan2(im, re);
    return a;
}

// komplex szám konjugáltja
public static ComplexPolar conjugate(ComplexPolar a) {
    return a.toPolar(a.re(), - a.im());
}
...
```

A komplex számok osztálya (ComplexPolar.java)

```
...
// komplex számok összeadása
public static ComplexPolar add(ComplexPolar a,
                               ComplexPolar b) {
    ComplexPolar c = new ComplexPolar();
    double re = a.re() + b.re();
    double im = a.im() + b.im();
    return c.toPolar(re, im);
}

// komplex számok kivonása
public static ComplexPolar subtract(ComplexPolar a,
                                    ComplexPolar b) {
    ComplexPolar c = new ComplexPolar();
    double re = a.re() - b.re();
    double im = a.im() - b.im();
    return c.toPolar(re, im);
}
...
```


A komplex számok osztálya (ComplexPolar.java)

```
...
// ket komplex szam szorzata
public static ComplexPolar multiply(ComplexPolar a,
                                   ComplexPolar b) {
    ComplexPolar c = new ComplexPolar();
    c.r = a.r * b.r;
    c.theta = a.theta + b.theta;
    return c;
}

// komplex szam n-dik hatvanya
public ComplexPolar powerN(ComplexPolar a, int n) {
    return a.toPolar(Math.pow(a.r,n) *
                     Math.cos(n * a.theta),
                     Math.pow(a.r,n) *
                     Math.sin(n * a.theta));
}
...
```

A komplex számok osztálya (ComplexPolar.java)

...

```
// a komplex szám sztring reprezentacioja
public String toString() {
    if (im() == 0) return re() + "";
    if (re() == 0) return im() + "i";
    if (im() < 0) return re() + "_-" + (-im()) + "i";
    return re() + "_+" + im() + "i";
}
}
```

Készítsük el a létrehozott osztály tesztprogramját! Ehhez adjunk meg két komplex számot! Számítsuk ki a két komplex szám összegét, az összeg konjugáltját (adjuk meg a konjugált modulusát, szögértékét fokban és radiánban (a szögértékek konverziójához hozzunk létre egy új osztályt (Angle.java))), a két komplex szám különbségét és szorzatát, valamint a szorzat n -dik (ahol n egy megadott nemnegatív szám) hatványát!

Fok \leftrightarrow radián konverziót végző osztály (Angle.java)

```
/**
 * Fok  $\leftrightarrow$  radian konverzio.
 */
public class Angle {

    // Fok  $\rightarrow$  radian konverzio
    static double degreeToRadian(double theta) {
        return Math.toRadians(theta);
    }

    // Radian  $\rightarrow$  fok konverzio
    static double radianToDegree(double theta) {
        return Math.toDegrees(theta);
    }
}
```

Tesztprogram (ComplexPolarTest.java)

```
/**
 * Komplex számok osztályának tesztelese.
 */

public class ComplexPolarTest {

    public static void main(String[] args) {
        ComplexPolar a = new ComplexPolar();
        a.r = 1.2;
        a.theta = Math.PI / 4;
        System.out.println("Az első komplex szám: "
+ a);
        ComplexPolar b = new ComplexPolar();
        b.r = 3;
        b.theta = Math.PI / 3;
        System.out.println("A második komplex szám: "
+ b);
        ...
    }
}
```

Tesztprogram (ComplexPolarTest.java)

```
...
    ComplexPolar c = new ComplexPolar();
    c = ComplexPolar.add(a,b);
    System.out.println("A_ket_szam_osszege:_"
+ c);
    c = ComplexPolar.conjugate(c);
    System.out.println
("A_ket_szam_osszegenek_konjugaltja:_"
+ c);
    System.out.println("A_konjugalt_modulusa:_"
+ c.r);
    System.out.println
("A_konjugalt_szogerteke_fokban:_"
+ Angle.radianToDegree(c.theta));
    System.out.println
("A_konjugalt_szogerteke_radianban:_"
+ c.theta);
...
```

Tesztprogram (ComplexPolarTest.java)

```
...
    c = ComplexPolar.subtract(a,b);
    System.out.println("A_ket_szam_kulonbsege:_"
+ c);
    c = ComplexPolar.multiply(a,b);
    System.out.println("A_ket_szam_szorzata:_"
+ c);
    int n = 5;
    c = c.powerN(c,n);
    System.out.println
        ("A_szorzat_" + n + "-dik_hatvanya:_"
        + c);
}
}
```

- Elegendő a főprogramot (`ComplexPolarTest.java`) lefordítani. A fordító ennek lefordítása közben ugyanis az összes többi hivatkozott osztályt is megpróbálja lefordítani.
 - Fordítás: `javac ComplexPolarTest.java`
 - Futtatás: `java ComplexPolarTest`

Csomag (package)

- A típusainkat csomagokba soroljuk
- Motiváció:
 - A programok fejlesztésének modularizálása
 - Névütközések feloldása névterek létrehozásával
 - Hozzáférés szabályozása
- A csomagok hierarchiába szervezhetők (csomagnevek hierarchiája)
 - Minősített hivatkozás, pontokkal elválasztva, pl. `java.util`
 - Konvenció: a csomagnevek globális szinten egyediek, a domain nevekhez hasonló felépítés szerint képződnek, pl. `hu.elte.geo`
- Egy típus teljes neve tartalmazza az őt befoglaló csomag nevét is, pl. `java.util.Vector`
- Egy típus pontosan egy csomagba tartozik
 - Csomagok metszete üres
 - Névtelen csomag (nincs `package` utasítás)

- Hivatkozás más csomagokra
 - Ha egy forrásfájlban használni akarunk egy típust egy másik csomagból (pl. névtelen csomagban, az A osztályt definiáló fájlban belül)
 - Hivatkozás teljes névvel (teljesen minősített név / fully qualified name)

```
class A {  
    void foo() {  
        java.util.Vector bar =  
            new java.util.Vector();  
    }  
}
```

- Importálás + rövid névvel történő hivatkozás

```
import java.util.Vector;  
class A {  
    void foo() { Vector bar = new Vector(); }  
}
```

Minden típust importálhatunk egyszerre a java.util csomagból: `import java.util.*;`

- Lépései:
 - A forrásszövegben elhelyezzük a befoglaló csomag azonosítóját (package + azonosító (teljesen minősített név))
 - A forráskódokat a csomag nevének megfelelő könyvtárhierarchiába szervezzük, a fordítást a munkakönyvtárból hajtjuk végre.

- Könyvtárszerkezet kialakítása + forráskódok elhelyezése:
 - `mkdir compol`
 - `mkdir compol\basics`
 - `mkdir compol\utils`
 - `mkdir main`
 - `move Angle.java compol\utils`
 - `move ComplexPolar.java compol\basics`
 - `move ComplexPolarTest.java main`

munkakönyvtár

- | - compol
 - | - basics
 - | ComplexPolar.java
 - | - utils
 - | Angle.java
- | - main
 - | ComplexPolarTest.java



Csomagazonosítók beillesztése a forrásszöveg elejére + importálás:

- Az Angle.java esetén: `package compol.utils;`
- A ComplexPolar.java esetén: `package compol.basics;`
- A ComplexPolarTest.java esetén: `package main, ill. import compol.utils.Angle;` és `import compol.basics.ComplexPolar;`
- Az osztályokat és a bennük szereplő összes adattagot és metódust `public` kulcsszóval kell ellátni.

- Fordítás: `javac main\ComplexPolarTest.java`
- Futtatás: `java main.ComplexPolarTest`

- javac -g HelloWorld.java
- jdb
- A jdb parancs legfontosabb opciói (l. help utasítással):

```
> help
** command list **
[.]
run [class args]      -- start execution of application's main class
[.]
print <expr>          -- print value of expression
[.]
eval <expr>          -- evaluate expression (same as print)
set <lvalue> = <expr> -- assign new value to field/variable/array element
locals                -- print all local variables in current stack frame
[.]
stop at <class id>:<line> -- set a breakpoint at a line
[.]
step                  -- execute current line
[.]
cont                 -- continue execution from breakpoint
[.]
exit (or quit)        -- exit debugger
[.]
```


Feladatok



- Javítsd ki a LINKen található HIBÁS programot.
- Point osztály:
 - egy kétdimenziós pontot valósít meg
 - a pontnak van egy x és egy y koordinátája
 - az osztálynak van egy statikus metódusa, amely két pontot kap paraméterül és visszaadja a két pont távolságát (`double`)
- Distance osztály:
 - az osztály tartalmaz főprogramot, amely a parancscsori paramétereket pontoknak értelmezi: a pontok szóközzel elválasztva vannak felsorolva, minden pontnál elől az x , utána az y koordináta (ezek is szóközzel elválasztva)
 - feltételezhetjük, hogy páros számú paraméter van, amelyek mind egyéssz számok
 - a program a Point osztály felhasználásával számítsa ki és adja össze az egymás mellett lévő pontok távolságát (pl. 3 pont esetén az 1. és a 2. pont távolságához hozzá kell adni a 2. és a 3. pont távolságát), majd az eredményt írja ki

- Példák:

```
> java Distance  
0.0
```

```
> java Distance 1 2  
0.0
```

```
> java Distance 0 0 3 4  
5.0
```

```
> java Distance 1 2 4 6  
5.0
```

```
> java Distance 1 2 4 6 7 6  
8.0
```

- Javítsd ki a programot!

- Készítsünk egy, a nemek ábrázolásához használt Gender nevű osztályt! Ebben szerepeljen két osztályszintű konstans, amelyek rendre Gender.MALE (férfi) és Gender.FEMALE (nő).
- Készítsünk Person névvel egy olyan osztályt, amelyben nyilvántartjuk a személyi adatokat! A rögzíteni kívánt adatok: a személy vezeték- és keresztneme (mindkettő String), foglalkozása (String), neme (Gender) és születési éve (int).
- Legyen a Person osztálynak egy olyan statikus metódusa makePerson() névvel, amely ezeket az adatokat paraméterként kapja és összeállít belőlük egy Person típusú objektumot! A létrehozás előtt azonban ellenőrizzük, hogy a születési év 1900 és 2017 közé esik-e. Ha nem, akkor a metódus üres, vagyis null referenciát ad vissza (azaz ilyenkor nem jön létre objektum)!

Személy osztály (Person.java)

- Egészítsük ki a `Person` osztályt egy `toString()` metódussal, amely `String` típusú értéké alakítja az adott objektum belső állapotát!
- Készítsünk egy `equals()` nevű metódust a `Person` osztályhoz, amely eldönti a paraméterként megadott másik `Person` objektumról, hogy megegyezik-e az aktuális példánnyal. Vigyázzunk arra, hogy mivel referenciát adunk át paraméterként, az lehet (többnyire véletlenül) null érték is! Ilyenkor értelemszerűen az eredménye hamis lesz.
- Tegyük az eddigi osztályokat a `person` csomagba és készítsünk hozzá egy főprogramot, amelyben létrehozunk két `Person` objektumot, megvizsgáljuk, hogy ugyanarról a két személyről van-e szó és az eredményt kiírjuk a szabványos kimenetre! A főprogram kerüljön a `main` csomagba!

Tribonacci-sorozat (Tribonacci.java)

Készítsünk programot (csomag nélkül, rekurzívan), amely előállítja a Tribonacci-sorozat tagjait az N . Tribonacci-számig, ahol N értékét parancssori paraméterként kérjük be! A Tribonacci-sorozatot a következő képlet generálja:

$$T_n = \begin{cases} 0, & \text{ha } n = 0; \\ 0, & \text{ha } n = 1; \\ 1, & \text{ha } n = 2; \\ T_{n-1} + T_{n-2} + T_{n-3}, & \text{ha } n \geq 3. \end{cases}$$

Euklideszi algoritmus (Euclid.java)

Készítsünk programot, amely az Euklideszi algoritmus alapján kiszámítja két szám (p és q) legnagyobb közös osztóját, ahol a számokat parancssori paramétereként kérjük be! Oldjuk meg a feladatot rekurzív és nem rekurzív függvénnyel is!

Készítsünk programot, amely kiszámítja egy másodfokú egyenlet gyökeit!
(A másodfokú egyenlet általános kanonikus alakja: $ax^2 + bx + c = 0$, ahol $a \neq 0$. A másodfokú egyenlet megoldóképlete:
$$x = -\frac{b}{2a} \pm \frac{\sqrt{b^2 - 4ac}}{2a} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
, diszkriminánsa: $D = b^2 - 4ac$. Ha valós együtthatós az egyenlet, akkor $D > 0$ esetén 2 valós, $D = 0$ esetén egy valós (kettős gyök), $D < 0$ esetén pedig 2 nem valós, komplex gyöke van.) Az egyenlet együtthatóit a billentyűzetről kérjük be (útmutatás: importáljuk a `java.util.Scanner` osztályt)!

Számológép (Calculator.java, Calculator2.java)

- Készítsünk egy `Calculator` nevű osztályt, amely két parancssori paramétert vár: az első paraméterben számok vannak vesszővel elválasztva, a második paraméter pedig egy szám (pl. `java Calculator 2,3,4 10`). Ellenőrizzük, hogy megfelelő számú paramétert kaptunk-e! Ha igen, akkor feltehetjük, hogy a paraméterek valóban számok. Konvertáljuk az első paraméterben megkapott számokat egész számokat tartalmazó tömbbé! Ezek után írjuk ki a tömb és a második paraméter összegét (minden elemet meg kell növelni a második paraméterrel).
- Segítség:
 - `public String[] split(String regex)`: a regex reguláris kifejezés (jelenleg a szövegrészeket elválasztó jel) mentén bontja fel a szöveget, majd a felbontott szöveget tömbként adja vissza.
 - új tömb létrehozása: `típus[] változónév = new típus[hossz];` pl.: `int[] t = new int[5];`

- Módosítsuk a Calculator osztályt a következőképpen: hozzunk létre két attribútumot, legyenek ezek `public static int[] a;` és `public static int b;`.
- Ezután definiáljuk a következő metódusokat:
 - `public static void loadInputs(String[] args):` nyers formában megkapja a parancssori paraméterek tömbjét, majd feltölti az argumentumokat,
 - `public static int[] add():` a függvény egy új tömbben tárolja el a megoldást, amelyet visszaad,
 - `public static void display(int[] result):` a kiíratást végző metódus.