Home

Last edited by **javagyak** 2 weeks ago

Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

Tantárgyi követelmények (http://kto.web.elte.hu/hu/oktatas/java/) Java SE 8 Documentation (http://docs.oracle.com/javase/8/docs/) Feladatok

Halló mindenki!

Írjuk meg Javaban a hagyományos "Hello world" programot!

```
/**
 * "Hello world" program.
 */

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

Fontos, hogy az állomány neve megegyezzen a benne definiált publikus osztály nevével. Vagyis, ha `Foo.java` az állomány neve, akkor legyen benne egy `Foo` nevű osztály, amelyet a `public class Foo` definícióval fejtünk ki.

Kiírás a konzolra

Hasonlóan más nyelvekhez, a konzolra itt is többféle módon lehet írni, a `java.lang.System` (http://download.oracle.com/javase/8/docs/api/java/lang/System.html) osztályon keresztül:

- Szabványos bemenet ("standard input"): `System.in` objektum.
- Szabványos kimenet ("standard output"): `System.out` objektum.
- Szabványos hibakimenet ("standard error"): `System.err` objektum.
- A kiírandó szövegben szerepelhetnek vezérlőkarakterek (escape sequence), pl. `r`, `n`, `t`, `b`, stb.
- A `System` osztály további segédműveleteket is tartalmaz, pl. `System.exit()`.

A környezet beállítása

A program lefordításához és futtatásához először gondoskodnunk kell a környezet beállításáról.

Windows esetén indítsuk el a parancssort, például a *Windows* + *R* gombkombináció lenyomása után a `cmd.exe` elindításával. Ezt követően a `PATH` környezeti változót kell módosítanunk:

```
$ PATH=%PATH%;"C:\Program Files\Java\jdk1.8.0_102-b14\bin"
$ echo %PATH%
...;C:\Program Files\Java\jdk1.8.0_102-b14\bin
```

Ekkor a következőnek működnie kell:

```
$ javac -version
javac  1.8.0_102-b14
```


GNU/Linux vagy UNIX rendszereken is lényegében ugyanezt kell megcsinálni, de ennek konkrét szintaxisa az adott shelltől függ. Illetve jobb esetben a csomagkezelő erről gondoskodik a JDK telepítésekor.

Fordítás

A programokat a `javac` ("Java compiler") segítségével tudjuk lefordítani.

```
$ javac HelloWorld.java
```

Amennyiben a program fordítási hibákat tartalmaz, úgy a fordító erről tájékoztat:

```
$ javac HelloWorldBad.java
HelloWorldBad.java:1: error: class, interface, or enum expected
public HelloWorldBad {
       ^
HelloWorldBad.java:2: error: class, interface, or enum expected
    public static void main(String[] args) {
                    ^
HelloWorldBad.java:4: error: class, interface, or enum expected
    }
    ^
3 errors
```

ahol a hibás kód ez volt:

```
public HelloWorldBad {
    public static void main(String[] args) {
         return 0;
    }
}
```

Futtatás

A sikeres fordítás eredményétől függően nulla vagy több `.class` kiterjesztésű (byte-kódot tartalmazó) állomány keletkezik. Ezeket az állományokat lehet futtatni a Java virtuális gép (Virtual Machine, VM), azaz a `java` segítségével.

```
$ java HelloWorld
Hello World!
```

Megjegyzés: A `java` csak olyan `.class` állományt tud futtatni, amelynek van publikus statikus `main()` metódusa (ld. a fenti példaprogramot)!

Egy olyan program, ahol nincs `main()` metódus:

```
public class HelloWorldWrong {}
```

Lefordul hiba nélkül:

Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

```
$ javac HelloWorldWrong.java
```

Azonban nem fog tudni futni:


```
$ java HelloWorldWrong
Error: Main method not found in class HelloWorldWrong, please define the main metho
   public static void main(String[] args)
```


Készítsünk el a `HelloWorld` egy olyan változatát, amely billentyűzetről bekér egy nevet és utána neki köszön.

Segítség:

- beolvasás billentyűzetről: `String name = System.console().readLine();`

Készítsünk el a `HelloWorld` egy olyan változatát, amely paraméterben kap egy nevet és utána neki köszön.

Segítség:

- a parancssori paramétereket az `args` változóban találjuk
- paraméterek darabszáma: `args.length`
- az első paraméter: `args[0]`

Dokumentáció generálása

A forráskódból, a megjegyzések segítségével a programhoz tartozó (API) dokumentáció készíthető a `javadoc` felhasználásával. Ehhez a programban speciális formátumú megjegyzéseket kell írni.

```
$ javadoc HelloWorld.java
Loading source file HelloWorld.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_77
Building tree for all the packages and classes...
Generating /HelloWorld.html...
Generating /package-frame.html...
Generating /package-summary.html...
Generating /package-tree.html...
Generating /constant-values.html...
Building index for all the packages and classes...
Generating /overview-tree.html...
Generating /index-all.html...
Generating /deprecated-list.html...
Building index for all classes...
Generating /allclasses-frame.html...
Generating /allclasses-noframe.html...
Generating /index.html...
Generating /help-doc.html...
```

Általános programozási konvenciók

```
package java.foo; // top-level domain (TLD), kisbetus karakterek

/**
 * osztalyleiras
 *
 * @version  0.1.0
 * @author   Mr. T
 */
public class Bar extends Baz {

    /** classVar1 egysoros comment. */
    public int classVar1;

    /**
     * classVar2, aminek meg tobbsoros
     * a leirasa.
     */
    private static String classVar2;

    /**
     * Konstruktor komment.
     */
    public Bar() {
        // ...
    }

    /**
     * Fuggveny komment.
     */
    public void doSomething() {
        // ...
    }

    /**
     * Valami masik fuggveny komment.
     *
     * @param someParam valami parameter
     * @return valami ertek
     */
    public int returnSomeValue(Object someParam) {
        // ...
    }

    /**
     * Logikai fuggveny.
     */
    public boolean isSomething() {
       // ...
    }
}
```


Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

A forráskódok elnevezésével, elrendezésével kapcsolatos konvenciók:

- Osztálynév = állománynév, nagybetűvel kezdődik,
- Csomagnév = könyvtárnév, kisbetűvel kezdődik, skatulyázható.

Megjegyzés: A Java ugyan megengedi az ékezetes karakterek használatát mind az állománynevekben, mind pedig az azonosítókban, azonban ez nem ajánlott!

Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

Típusok

A nyelvben találhatóak beépített primitív típusok, néhány közülük: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`. Ezek rendelkeznek kezdőértékkel, amely a típustól függ (pl. `0` az `int` típus esetén, `false` a `boolean` esetén). Értékeik megadhatóak oktális (`int octVal = 01`), hexadecimális (`byte hexVal = 0xff`), vagy tudományos (`double d = 1.23e4`) jelöléssel.

A többi típust objektumosztályok írják le, illetve szükség esetén (akár automatikusan) a primitív típusú értékek maguk is objektumokba csomagolhatóak (`java.lang.Byte` (http://download.oracle.com/javase/8/docs/api/java/lang/Byte.html), `java.lang.Short` (http://download.oracle.com/javase/8/docs/api/java/lang/Char.html), `java.lang.Integer` (http://download.oracle.com/javase/8/docs/api/java/lang/Integer.html), stb.). Továbbá számos, a típusokhoz tartozó műveletet a megfelelő osztály statikus függvényeként implementálták.

Típusok közti konverzió:

- bővítő konverzió, a fordító automatikusan engedi,
- szűkítő konverzió, típuskényszerítéssel (`byte b = (byte) 300`) kérhető.

Konverzió szöveggel:

- Szöveget a `java.lang.String` (http://download.oracle.com/javase/8/docs/api/java/lang/String.html) osztállyal lehet ábrázolni.
- Számok szöveggé alakítása: `String s = "" + 1;`
- Szöveg számmá alakítása: `Integer.parseInt("1")`, `Double.parseDouble("2.0")`, stb.

Vezérlési szerkezetek

A nyelv jellegében imperatív vezérlési szerkezeteket tartalmaz, amelyek mind utasítások. Lentebb találhatóak a fontosabb kategóriák. Ezeken kívül még a nyelv természetesen támogatja több utasítás összefogását egy blokkba. A blokkokat a `{` és `}` jelek határolják.

`if`, `else`

```
if ( /* elso feltetel */ ) {
    // ha az elso feltetel igaz
} else
if ( /* masodik feltetel */ ) {
    // ha a masodik feltetel igaz
} else
if ( /* harmadik feltetel */ ) {
    // ha a harmadik feltetel igaz
} else {
    // egyebkent
}
```

`switch`

A `switch` utasítással többágú elágazásokat hozhatunk létre, azonban nem feltételek, hanem egy kifejezés értéke alapján. Ennek típusa integrális, azaz `byte`, `short`, `char`, `int` vagy ezek csomagoló osztálya (`Byte`, `Short`, `Character`, `Integer`) lehet, JDK 7 óta lehet String is.


```
int month = 8;

switch (month) {
    case 1:  System.out.println("Jan"); break;
    case 2:  System.out.println("Feb"); break;
    case 3:  System.out.println("Mar"); break;
    case 4:
    case 5:
    case 6:  System.out.println("Apr, May or Jun"); break;
    default: System.out.println("Other month"); break;
}
```

Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

`for`, `while`, `do`

```
while ( /* logikai feltetel */ ) {
    // ciklusmag
}
```

```
do {
    // ciklusmag
} while ( /* logikai feltetel */ );
```

```
for (/* inicializalas */; /* megallasi feltetel */; /* leptetes */) {
    // ciklusmag
}
```

```
for (;;) {  // vegtelen ciklus
    // ciklusmag
}
```

```
for (String a : args) {  // tombokre, iteralhato adatszerkezetekre
    // ciklusmag
}
```

`break`, `continue`, `return`

A `break` utasítás segítségével kiléphetünk blokkok, például ciklusmagok belsejéből. Ilyenkor a vezérlés a blokkot követő utasításra kerül.

```
for (;;) {
    if (false)
        break;
}
```

A `continue` utasítással a ciklusmag fennmaradó részét ugorhatjuk át, vagyis használatakor a ciklus újrakezdődik és a vezérlés a feltétel kiértékeléséhez kerül.

```
for (int i = 0; i < 10; ++i) {
    if (i < 9)
        continue;
    System.out.println("i = " + i);
}
```


Megjegyzés: Létezik a nyelvben a `goto` kulcsszó, de nem használható. Ellenben a `break` és `continue` paraméterezhetők címkékkel.

A `return` segítségével léphetünk vissza alprogramokból (metódusokból). Ha van visszatérési érték, akkor itt adhatjuk meg.

```
static void main(String args) {
    if (args.length == 0)
        return;
    System.out.println("Never reached.");
}

static int succ(int i) {
    return (i + 1);
}
```

Valós számok

A valós számok (a `float` és `double` típusok) alkalmazása alapos körültekintést igényel -- nem csak a Java nyelvben, de általában minden más nyelven is. Ennek oka, hogy a gépen tárolt valós számok csak közelítések, amelyek fixpontos vagy lebegőpontos ábrázolásúak lehetnek.

- Fixpontos ábrázolás esetén az előjelet, az egészrészt és a törtrészt adott mennyiségű biten tárolják, függetlenül a konkrét értéktől. Ez így elég nagy információvesztéssel is járhat, viszont gyors.
- Lebegőpontos ábrázolás esetén az értékeket egy ún. normálalakban tárolják.

Példa a reprezentációval járó numerikus hibára:

```
System.out.println(0.2 + 0.2 + 0.2 + 0.2 + 0.2);  // kis numerikus hiba: 1.0
System.out.println(0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f + 0.1f)
System.out.println(0.1f + 0.1d + 0.1d + 0.1d + 0.1d + 0.1d + 0.1d + 0.1d + 0.1d + 0.1d)
```

Az == operátor

A fentiek miatt valós számok közvetlen összehasonlítása, különösen ciklusok feltételeiben, nem ajánlott.

```
System.out.println(0.3 == 0.1d + 0.1d + 0.1d);
```

vagy

```
for (double d = 0.0; d != 0.3; d += 0.1) {
    // Vegtelen ciklus lesz!
}
```

Ezt a hiányosságot ún. hibahatár definiálásával lehet kiküszöbölni. Vagyis két valós számot megegyezőnek tekint, amennyiben azok egy adott epsilon távolságon belül találhatóak.

```
double delta = 1.0E-5; // hibahatar (epszilon)
double d1    = 0.3
double d2    = 0.1 + 0.1 + 0.1;

if (Math.abs(d1 - d2) < delta) {
    System.out.println("d1 == d2");
}
```


Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

Megjegyzés: Az `abs()` metódus a `java.lang.Math` (http://download.oracle.com/javase/8/docs/api/java/lang/Math.html) osztály eleme, ahol további matematikai függvények is találhatóak.

Túl- és alulcsordulás

A gépi ábrázolás korlátai miatt minden számtípus rendelkezik egy minimális és maximális értékkel (például `Integer.MIN_VALUE` és `Integer.MAX_VALUE`), amelyek átlépésével alul- vagy túlcsordulás következik be. Ez nehezen észlelhető hibákhoz tud vezetni.

```
double big = 1.0e307 * 2000 / 2000;
System.out.println(big == 1.0e307);
```

Amikor beszorozzuk a számot, kimegyünk a `double` típus ábrázolható tartományából, és az `Infinity` értéket kapjuk, amelyet újból elosztva már nem az eredeti számot kapjuk vissza!

```
System.out.println(1234.0d + 1.0e-13d == 1234.0d);
```

Ha kiírunk a konzolra egy valós számot, akkor a megjelenő érték nem a reprezentációban használt közelített érték lesz.

```
System.out.println(0.1d);  // Megjeleno ertek: 0.1
```

viszont:

```
System.out.println(0.1 == 0.09999999999999998); // hamis
System.out.println(0.1 == 0.09999999999999999); // igaz
System.out.println(0.1 == 0.10000000000000001); // igaz
```

A közelített érték egy speciális osztály segítségével jeleníthető meg:

```
// A kiirt ertek: 0.1000000000000000055511151231257827021181583404541015625
System.out.println(new BigDecimal(0.1));
```

Tömbök

Minden `T` típushoz van `[]`, vagyis tömb típus. A tömbök objektumok, ezért referenciákkal hivatkozunk rájuk, valamint vannak adattagjaik. Emiatt a tömbök mindig ismerik a saját hosszukat, valamint külön példányosítani kell ezeket.

```
int[] array1 = new int[5];
int array2[];
int array3[] = { 1, 2, 3, 4, 5 };
```



```
for (int i = 0; i < array3.length; ++i) {
    System.out.println(array3[i]);
}
```

Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

Többdimenziós tömbök és létrehozhatóak a `[]` jelölés ismételt alkalmazásával. A példányosításkor az első dimenziót legalább meg kell adni.

Vesd össze:

```
int[][] multidim = new int[5][5];
```

vagy

```
int[][] multidim = new int[5][];

for (int i = 0; i < multidim.length; i++) {
    multidim[i] = new int[i + 1];
}
```

Operátorok

A nyelvben használhatóak a C-szerű nyelvekben már látott operátorok: `==`, `!=`, `&&`, `||`, `+`, `-`, `*`, `/`, `%`, `++`, `--`, `+=`, `-=`, `*=`, `/=`, `?:`, stb.

Az operátorok eredményének típusa mindig a bővebb paraméter típusa lesz, de legalább `int`, például:

```
double d = 1 / 2;  // eredmenye: 0.0
byte b = 1 + 2;    // explicit tipuskenyszerites kell
```

Prefix és postfix operátorok

A `++` operátort lehet prefix és postfix jelöléssel is alkalmazni, ez viszont befolyásolja a viselkedését is.

```
int i = 0;
System.out.println(i++); // kiir, megnovel: "0"
System.out.println(++i); // megnovel, kiir: "2"
```

Mi az eredménye (vö. C++)?

```
int i = 0;
System.out.println("" + i++ + ++i); // C++: architekturafuggo
```

Mi lesz az eredménye?

```
int i=0;

i=i++;
i=i++;
i=++i;
System.out.println(i);
```


Clone repository (/javagyak/01/wikis/git_access)

Home (/javagyak/01/wikis/home)

Objektumok összehasonlítása

Objektumokat mindig az `equals()` metódussal tudunk összehasonlítani, az `==` operátor csak referencia szerinti összehasonlítást végez.

Például szövegeket is ezen a módon hasonlíthatunk össze:

```
boolean b1 = "a" == "a";       // lehet hamis!
boolean b2 = "a".equals("a");  // mindig igazat kell adnia
```

Összehasonlító operátor feltételekben

Amikor feltételekben adunk hasonlítunk össze kifejezéseket, akkor lehetőség szerint bal oldalra írjuk a konstansokat!

```
boolean b = false;

if (true == b) {
    // ...
}
```

Feladatok

- Készítsünk egy olyan programot, amely két szám paramétert vár, majd kiírja ezek összegét, különbségét, szorzatát és - amennyiben a 2. szám nem nulla - a hányadosát.

Segítség: - egy `String` számmá alakítása: `int n = Integer.parseInt("32");` - elágazás: `if (...) { ... }` vagy `if (...) { ... } else { ... }` - osztás: `/`, maradék: `%`

- Egészítsük ki az osztályt a hatványozás műveletével.

Segítség: - Ciklus: `for (int i = 0; i < limit; i++) { ... }` vagy `while (...) { ... }`

- A hatványozás működjön negatív számokra is.

Segítség: - Ehhez használjunk lebegőpontos típust: `double exp = 1.0;`

- Készítsünk egy hőmérsékleti skálák közt konvertáló programot! Ehhez olvassunk be egy számot és karaktert a szabványos bemenetről: ha karakter `c` akkor a számot Celsiusról Fahrenheit fokra számoljuk át, ellenkező esetben Fahrenheitről Celsius értékre számoljunk.
- Írjuk meg az `n` faktoriálisát kiszámoló programot.
- Írjuk meg az `n`. fibonacci számot kiszámoló programot. fibonaci számok: `1 1 2 3 5 8 ...`
- Készítsünk egy olyan programot, ami képes megkeresni egy (pozitív egész) szám négyzetgyökét. A módszer legyen a következő: Ha `x` a szám, aminek a négyzetgyökét keressük, akkor kezdetben tudjuk, hogy ez `0` és `x` közé esik. Felezzük meg a `[0,x]` intervallumot, és vizsgáljuk meg, hogy a középső érték

négyzete mekkora. Ha nagyobb `x` -nél, akkor a jelenlegi intervallum felső határát változtassuk a jelenlegi középpontra, ha kisebb, akkor pedig az alsót. Mindezt addig folytassuk, amíg egy megadott pontossággal meg nem közelítettük a négyzetgyököt.

- Valósítsunk meg egy minimális konzolos számológépet! Olvassuk be az operátort, majd annak az operandusait és írassuk ki a konzolra a számolás eredményét! Támogassuk a következő operátorokat: összeadás (+), kivonás (-), szorzás (*) és osztás (/). Minden egyéb esetben adjunk hibajelzést és kérjünk be új adatokat.
- Írjunk egy olyan programot, amely egy beolvasott számra eldönti, hogy tökéletes-e! Tökéletesnek nevezünk olyan egész számokat, amelyek megegyeznek az osztóik (az egyet beleértve, önmagukat kivéve) összegével. Az első négy ilyen szám a 6, 28, 496, 8128.
- Az előző megoldásunkat egészítsük ki úgy, hogy egytől a paraméterként megadott határig minden egész számra vizsgálja a tökéletességet és megadja, hogy abban az intervallumban mennyi ilyet talált! Ha nem talált egyetlen ilyen számot sem, akkor írja ki, hogy `Egyetlen tökéletes szám sincs a megadott intervallumban.`
- írjunk olyan programot, amely előállítja a Collatz-sorozat tagjait!

Clone repository (/javagyak/01/wikis/git_access)
Home (/javagyak/01/wikis/home)

Linkek

Kapcsolódó forráskódok (http://oktnb127.inf.elte.hu/javagyak/01/tree/master)