2. Gyakorlat

legendi@inf.elte.hu

2010. február 17.

Kiegészíés

for-each ciklusok Parancssori argumentumok listázása:

```
foreach (String act : args) {
    System.out.println(act);
}
```

Tetszőleges tömbre.

Escape sequences $'\backslash r', '\backslash n', '\backslash t', '\backslash b'$, etc.
Részletesen: `http://java.sun.com/docs/books/tutorial/java/data/characters.html`

Környezet beállítása Konzolban:

```
Microsoft Windows XP [verziószám: 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\tmp>set PATH=%PATH%;c:\Program Files\Java\jdk1.6.0_12\bin\

c:\tmp>javac -version
javac 1.6.0_12

c:\tmp>javac HelloWorldApp.java

c:\tmp>java HelloWorldApp
Hello World!

c:\tmp>
```

Operátorok Eredmény típusa *mindig* a bővebb paraméter típusa (`double d = 1 / 2;` eredménye `0.0` lesz!).

Stringek összehasonlítása Mint az objektumokat: `equals()` metódussal (az == operátor referencia szerinti összehasonlítást végez csak, nem tartalom szerintit).

```
boolean b1 = "a" == "a";       // lehet hamis!
boolean b2 = "a".equals("a"); // mindig megfeleloen mukodik
```

Összehasonlító operátor Baloldalra lehetőleg konstanst írjunk. C++ probléma itt nem lehet, mert 0, 1 nem szerepelhet elágazás, ciklus terminálási feltételében, de kellemetlen helyzetek így is adódhatnak:

```
boolean b = false;

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

Igyekezzunk baloldalra konstansokat írni.

Többdimenziós tömbok Inicializálásnál az 1. dimenziót kell csak megadni, azonban ha tudod a többit, azt is érdemes:

```
int[][] arr = new int[9][9];
```

Java forrás Egy (jó) része nyílt, a forrás megtalálható a JDK könyvtárában (`src.zip` fájl).

Linkek

A tárgy honlapja Kozsik Tamás oldalán: `http://aszt.inf.elte.hu/~kto/teaching/java/`

Java Language Specification Harmadik kiadás, rendes specifikáció, HTML, PDF formátumban. `http://java.sun.com/docs/books/jls/`

Csomagok

Modularizáció, névütközések feloldása, hozzáférés szabályozás, etc. Osztályok, interfészek gyűjteménye. Használható a `*` wildcard. Alapértelmezetten látszik a `java.lang.*` csomag minden eleme, minden mást importálni kell (anélkül ún. *fully qualified classname*-mel hivatkozhatunk, pl. `java.util.Vector`):

```
import java.util.Vector; // 1 tipushoz
import java.math.*;      // Minden package-beli tipus lathatova valik

import java.awt.*;         // GUI
import java.awt.event.*;   // GUI - esemenykezeles
import javax.swing.*;      // Advancedebb GUI
import java.util.*;        // Adatstrukturak
import java.io.*;          // IO
import java.util.regex.*;  // Regexp

// static import: minden static konstans lathato az adott osztalybol
// fenntartasokkal hasznalni
import static java.lang.Math.*;
```

A fordítás nehézkes, nem működik a `*.java`.... Minden .-tal szeparált rész egy könyvtárat jelent, fordítás a gyökérkönyvtárból. Static importot csak offtosan (struktúráltság, enkapszuláció, egysébge zárás ellen hat - használjatok helyette dedikált osztályt vagy interfészt). Csomag definíciója a Java fájl legelején:

```
package pkg;

// Import utasitasok

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

Fordítás teljes útvonal megadásával:

```
C:\tmp>javac pkg/*.java

C:\tmp>java pkg.HelloWorldApp
Hello World!

C:\tmp>
```

Ha esetleg nevutkozes van (2 ugyanolyan osztaly van importalva), akkor minosetet nevvel erhetjuk el mindkettőt. Importokat hasznaljatok nyugodtan, nem gaz, nem emeszt eroforrast (nem c++, dinamikus).

Feladat

Hozzatok létre egy `gyak2` csomagot, majd készítsetek egy DirLister alkalmazást! Ez parancssori paraméterként egy teljes elérési utat várjon, és hozzon létre egy java.io.File objektumot (`File dir = new File(args[0]);`)! Ezután listázzátok ki a könyvtárban található összes, ".java"-ra végződő fájlt (String#endsWith(), File#listFiles())!

Függvények

Általános prototípus:

```
<módosítószavak> <visszatérési érték> <név>( <paraméterek listája> )
        [ throws <kivétel lista> ] {
    <utasítás1>;
    <utasítás2>;
    ...
}
```

- Módosítószavak:
  - Láthatóság: public, protected, private. Ha nem definiált, akkor ún. package-private láthatóság.
  - Lehet abstract - leszármazottban kötelezően felüldefiniálandó
  - Lehet final - felüldefiniálhatóság letiltására
  - Lehet static - osztály szintű függvény (megjegyzés: static kontextusból csak static módosítóval ellátott hivatkozás szerepelhet)
  - Egyéb, pl. strictfp, native, synchronized, transient, volatile (utóbbi kettő **csak** fieldekre). Ezekről később.
- Visszatérési érték szerinti csoportosítás:
  - `void`: eljárás
  - Minden egyéb: függvény
- Metódusnév: lowerCamelCase
- Paraméterátadás: minden paraméter érték szerint adódik át, *még a referenciák is.*

Szignatúra A függvény neve és paramétereinek típusa – más **nem**. Például:

```
eredmenyMeghatarozasa( double, int, int)
```

Overloading, overriding.

Feladat

Készítsetek egy függvényt, amely az Euklideszi-algoritmus alapján meghatározza két szám legnagyobb közös osztóját! A pszeudokód:

```
function gcd(a, b)
    if a = 0
       return b
    while b ? 0
        if a > b
           a := a - b
        else
           b := b - a
    return a
```

Kivételek

Általános forma:

```
try {
    ... // Kritikus utasitasok
} catch (Exception1 e1) {
    ...
} catch (Exception2 e2) {
    ...
} finally {

}
```

Az első ág, amelybe a kivétel osztályhierarchia szerint beleillik, lekezeli. Újradobás lehetséges: *throw e1;*, etc.

Alapvetően három típusú kivétel:

1. Felügyelt kivételek – definiálni kell őket a függvényben, és ha definiáltak, le is kell őket kezelni (ősosztály: `java.lang.Exception`, pl. `java.lang.ClassNotFoundException`, `CloneNotSupportedException`)
2. Felügyeletlen kivételek – nem kötelező sem difiniálni, sem lekezelni őket (ősosztály: `java.lang.RuntimeException`, pl. `ArrayIndexOutOfBoundsException`, `NumberFormatException`, `DivisionByZeroException`)
3. Léteznek még `Error`-ok, ezek a `Throwable` leszármazottai. Kritikus esetben fordulnak elő, a lekezelésük is felesleges a legtöbb esetben (pl. `OutOfMemoryError`, `StackOverflowError`)

Mindkettő őse a `Throwable` osztály - ezt kell hát lekezelni, ha mindent lehetőségre fel akarunk készülni. Jó tanács: üres kivételkezelő blokkot **soha** ne készítsünk! Legegyszerűbb megoldás: `e.printStackTrace()`. További funkciók az Exception osztály javadoc-jában. Kivételt dobni a `throw` utasítással lehet.

Példák

Egyszerű kivételkezelés

```
public static void main(String[] args) {
    try {
        int res = Integer.parseInt(args[0]);
        // ...
    } catch (NumberFormatException nfe) {
        System.err.println("Hibas input: " + args[0]);
        nfe.printStacktrace();
    }
}
```

Függvénydefiníció

```
static double divide(int a, int b) trhows Exception {
    if (0 == b) {
        throw new RuntimeException("0-val valo osztas!");
    }

    return (double) a / b;
}

public static void main(String[] args) {
    try {
        double res = divide(1, 0);
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Részletesen: `http://java.sun.com/docs/books/tutorial/essential/exceptions/`
Saját exception is definiálható, csak származtatni kell (pl. a `java.lang.RuntimeException`, `java.lang.Exception` osztályokból).

Feladat

Készítsetek egy függvényt, amely megadja egy másodfokú egyenlet gyökeit! A függvény definíciója legyen a következő:

```
private static int[] sqroots(final int a, final int b, final int c)
        throws Exception {
    // ...
}
```

A függvény dobjon kivételt, ha $a == 0$, vagy a diszkrimináns negatív! A függvény által dobott kivételeket kezeld is le a `main()` függvényben! A paramétereket a parancsori argumentumok határozzák meg, és az `Integer.parseInt()` függvény által dobott `NumberFormatException` kivételt is kezeljétek le ugyanabban a kivételkezelő ágban!

Objektumok

Minden típus az Object leszármazottja, ha nincs közvetlen őse. Javaban nincs többszörös öröklődés (kivéve interfészeknél). Fontosabb függvények:

equals(Object) Azonosság vizsgálat, contract szerint reflexív, tranzitív, szimmetrikus reláció, valamint konzisztens, és `x.equals(null)` értéke mindig hamis legyen.

hashCode() Hasheléshez (pl. egyes set implementációk, vagy a Hashtable esetén). Contract szerint:

- Ugyanarra az objektumra hívva konzisztens értéket ad (ha az objektum nem változik, ugyanazt az értéket adja).
- Ha két objektum az equals() szerint megegyezik, a hashCode() is egyezzen meg.
- Két különböző objektumra *nem kell* különböző értéket adni (ld. *hash collision*).

Példa:

```
private int i = 0;
private String str = "str";
private boolean b = false;

@Override
public int hashCode() {
final int prime = 31;
int result = 1;
```

```
result = prime * result + (b
? 1231
: 1237);
result = prime * result + i;
result = prime * result + ((str == null)
? 0
: str.hashCode());
return result;
}
```

toString() Az objektum szöveges reprezentációját adja vissza (Stringként).

Egyéb függvények clone(), finalize(), notify(), wait(), getClass()

Osztályok definiálhatók azonos forrásállományban (de csak egy public lehet), másik osztályban (belső osztályok), függvényen belül (lokális osztályok). Például:

```
package gyak2;

class A { ... }

public class B {
    ...
    class C { ... }

    void parse() {
        class D { ... }
        ...
    }
}
```

Beágyazott osztályok lehetnek static-ok (nincs szükség a befoglaló osztály egy példányára). Ha egy osztály final, nem származtatható. Ha strictfp, akkor minden művelete strictfp.

Származtatás

Javaban csak egyszeres öröklődés van, kivéve az interfészek esetében. Általános forma:

```
class Foo extends Bar implements Baz1, Baz2, ... {
    ...
}
```

A felüldefiniált függvények elé írjuk oda az `@Override` annotációt, es segít a compilernek kiszűrni az elgépelésből adódó problémákat fordítási időben. Adattagok elérése: getter/setter függvények.

```
@Override
public String toString() { ... }
```

Konstruktorban a szülőre a super(), az aktuális példány valameny konstruktorára this() hívással hivatkozhatunk. Ha ezeknek a paramétere egy függvény visszatérési értéke lehet, az csak statikus függvény lehet.

```
class A {
    protected int size;

    public A(int size) {
        this.size = size;
    }
}

class B extends A {
    public B()         { this(0); }
    public B(int size) { super(size); }
}
```

Származtatásnál a szűkebb hatókör nem megengedett. Visszatérési értékre nem lehet túlterhelni, mert az nem része a szignatúrának. Visszatérési értéket lehet specializáli (hasznos pl. a clone() függvénynél - kovariáns kötés). A 'this' pszeudováltozó.

Létrehozás, életciklus

Objektum létrehozása a new operátorral történik:

```
A a = new A(5); // Konstruktorhívás
```

Felszabadítással nem kell foglalkozni, azt megoldja a GC (finalize). Memória: dinamikus/statikus/stack (utóbbihoz nem fértek hozzá), automatikusan felügyelt (eden, etc.), `System.gc()`, `finalize()`. A Java által használható memóriaméret megadható a -Xmx, -Xms, etc. paraméterekkel. Az aktuális értékek lekérdezhetők a `Runtime` osztály metódusaival

Statikus/dinamikus típus: statikus, amivel definiálva lett, dinamikus, amilyen referenciára éppen mutat.

```
A a = new B(5);
```

A fenti példában az `a` változó statikus típusa A, dinamikus típusa B (ilyesmi megengedett, altípusos polimorfizmus, ld. Liskov-féle szubsztitúciós elv). A dinamikus típus leellenőrizhető az instanceof operátorral:

```
if (a instanceof B) { ... }
else if (a instanceof C) { ... }
```

elfedés

Részletek: `http://java.sun.com/javase/6/docs/api/java/lang/Object.html`

Feladatok

Mátrixok

Valósítsd meg a mátrixok típusát Javaban! Két konstruktor legyen: az egyik a méretet adja meg, és minden elem 0 legyen, a másik pedig egy double[][] paramétert kapjon! Valósítsd meg az általános függvényeket (equals(), hashCode(), toString()), valamint mátrixokat lehessen összeadni, és adott valós számmal beszorozni, valamint lekérdezni, hogy négyzetes-e.

Számok megvalósítása

Valósítsd meg a komplex és racionális számok típusát! A két osztály közös őse legyen a `MyNumber` osztály! A megvalósítandó műveletek: összeadás, kivonás, szorzás, valamint a komplex számok esetén kérdezhessük még le azok hosszát ($|a + bi| == \sqrt{a^2 + b^2}$).

Fájl írása

```
package gyak2.pm;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class WriteFile {
    public static void main(String[] args) {
        PrintWriter pw = null;

        try {
            pw = new PrintWriter(args[0]);

            pw.println("Line1");
            pw.println("Line2");
```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {
            if (pw != null) pw.close();
        }
    }
}
```

Fájl olvasása

```
package gyak2;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Hianyzik a filenev!");
            System.exit(1);
        }

        BufferedReader br = null;

        try {
            br = new BufferedReader(new FileReader(args[0]));
            String line = null;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (br != null) {
                try { br.close(); } catch (IOException e) { e.printStackTrace(); }
            }
        }
    }
}
```

+/- Feladatok

A feladatokat a `legendi@inf.elte.hu` címre küldjétek, **szombat éjfélig**! A subject a következőképp nézzen ki:

```
csop<csoportszám>_<EHA-kód>_<a megoldott feladatok száma>
```

Például:

```
csop1_LERIAAT_1  (csak az első feladattal)
csop1_LERIAAT_12 (mindkét feladattal)
```

Mellékelni csak a Java forrásfájlokat mellékeljétek (semmiképp ne teljes Eclipse/NetBeans projecteket), és egy levelet küldjetek (több levél esetén az utolsó mellékleteit értékelem)! Ajánlott az első feladatot NetBeans, a másikat Eclipse alatt készíteni. Szokni kell a környezetet, debuggolást. További fontos kritériumok:

- Ékezetes karaktereket **ne** használjatok! Főleg azonosítók esetében ne! A Java ugyan ezt megengedi, ugyanakkor a különböző környezetekbe való konvertáláskor ($latin2 \leftrightarrow UTF-8 \leftrightarrow Cp1250$) összetörnek a karakterek! Az ilyen forrásokat fordítani, következésképp értékelni sem tudom.

1. Feladat

Készítsetek egy minimális alkalmazott nyilvántartó alkalmazást. Minden alkalmazottnak van neve és éves fizetése. A cégnél kétféle alkalmazott létezik: junior és senior (ezek külön osztályok legyenek!), előbbiből maximum 10, utóbbiból maximum 3 fő dolgozhat a cégnél. Ezeket az adatokat tárold tömbben. Junior kollégák a fizetésük mellé étkezési utalványt kapnak, ennek az éves mértékét is szeretnénk tárolni, a senior kollégák pedig részvényeket szereznek a cégben.

Az alkalmazás egy konzolos menün keresztül az alábbi funkciókat nyújtsa (használjátok a Console osztály readLine() metódusát!):

- Új alkalmazott felvétele (amennyiben van rá lehetőség)
- Jelenlegi alkalmazottak listázása
- Adott alkalmazott adatainak módosítása
- Kilépés

A megoldás érdekében definiáld felül a toString(), hashCode() és equals() függvényeket! Az egyes osztályok külön fordítási egységekben legyenek. A Main osztály a gyak2.pm1 csomagban, az összes többi a gyak2.pm1.data csomagban!

2. Feladat

Készítsünk egy Sudoku ellenőrző programot! A program inputja egy fájl, amelynek soraiban pontosan 81 karakter található, és egy-egy Sudoku sorfolytonos ábrázolását jelenti. A program outputja egy fájl legyen, amelyben az érvényes kitöltést tartalmazó sorok kerülnek bele!