3. Gyakorlat

legendi@inf.elte.hu

2010. február 23.

Kiegészítés

Init blokkok Osztály hivatkozásakor, példányosításkor.

```
public class InitTest {
    static {
        // static tagokkal egyidoben,
        System.out.println("1");
    }

    {
        // osztaly tagokkal egyidoben ertekelodik ki
        System.out.println("2");
    }

    public InitTest() {
        System.out.println("3");
    }

    public static void main(String[] args) {
        // Visszateresi ertekkel nem kell foglalkozni
        new InitTest();
        System.out.println("4");
    }
}
```

Megkötések:

- Nincs forward referencing
- Nincs return (reccs)
- Static nem dobhat ellenőrzött kivételt
- Példányinicializátor dobhat ellenőrzött kivételt, ha *minden* konstruktor dobja azt a kivételt.

finalize() Nem destruktor!

equals() overloading vs. overriding, ld. FAQ!

String - immutable! Párhuzamosság miatt fontos:

```
String string = "AAAxAAA";

string.replace('x', 'A');

System.out.println(string); // "AAAxAAA"
string = string.replace('x', 'A');
System.out.println(string); // "AAAAAAA"
```

Vagy StringBuilder, StringBuffer használható:

```
StringBuffer sb = new StringBuffer();
sb.append("Hello ").append("World");
sb.reverse();
System.out.println( sb.toString() ); // "dlroW olleH"
sb.reverse();
sb.setCharAt(6, '-');
System.out.println( sb.toString() ); // "Hello-World"
sb.deleteCharAt(6);
System.out.println( sb.toString() ); // "HelloWorld"
sb.delete(0, sb.length() );
System.out.println( sb.toString() ); // ""
```

Részletesen: `http://java.sun.com/javase/6/docs/api/java/lang/StringBuilder.html`

catch Finally, catch opcionális, pl.:

```
try {
    ...
} finally {
    ...
}

try {
    ....
} catch (Throwable t) {
    ....
}
```

Típuskonverzió A felső bitek esnek ki, pl.:

```
int i = 0x12345678;
short s = (short) i; // 0x45678
byte b = (byte) i;   // 0x78
```

Labelek Használható break, continue esetén:

```
LABEL_OUTER:
for (int i=0; i<10; ++i) {
    for (int j=0; j<10; ++j) {
        if ( ... ) {
            break LABEL_OUTER;
        } else if ( ... ) {
            continue LABEL_OUTER;
        }
    }
}
```

Csomagok Leképezés fájlrendszerre.

Interfészek

Részletesen: `http://java.sun.com/docs/books/tutorial/java/IandI/index.html`

Új referencia típus, absztrakt függvények és konstansok gyűjteménye. Absztrakciós szintet vezet be, felületet definiál. Osztály megvalósít egy interfészt (`implements`), ha minden függvényét megvalósítja (`abstract` osztálynál nem kötelező, ugye).

Eltérés az osztályoktól:

- Többszörös öröklődés (névütközésre figyelni, függvényekre fordítási hiba lesz, nem C++, konstansok minősített névvel elérhetők). Szépen ezt úgy mondják, hogy a specifikáció többszörösen örökölhető, kód csak egyszeresen.
- Nincs közös ős (mint osztályoknál az `Object`)

Deklaráció

Mint az osztályoké:

```
interface A {}

public interface B {}
```

Öröklődési reláció neve itt kiterjesztés, lehet többszörös:

```
interface C extends A, B {}
```

Körkörös kiterjesztés → reccs.

Tagok

Mint az osztályoké, de:

- minden adattag public, static és final alapból
- minden függvény public és abstract alapból (ezeket nem kell kiírni se). Más nem lehet.

Előbbiből következik, hogy minden adattagot inicializálni kell (különben reccs), és ez csak már ismert érték lehet (forward referencing tilos):

```
interface D {
    int I = J; // Hibas definicio!
    int J = 0;
}
```

Nem szerepelhet this, super sem. Módosítószavak között nem szerepelhet synchronized, transient, volatile - ezek olyan dolgokat kötnek meg, amiknek implementációs szinten kell eldőlniük, használatuk ésszerűtlen lenne (?).

Megjegyzés Statikus típus: legyen minél kevésbé speciális (megvalósítás könnyen lecserélhető). (. . .)

Példa

```
interface RideableStuff {
    public final static int MAX_SPEED = 90;
    public abstract int getSpeed();
}

interface ColoredStuff {
    public abstract int getColor() throws UndefinedColorException;
}

class Pony implements ColoredStuff, RideableStuff {
    public int getSpeed() { return MAX_SPEED / 10; }

    public int getColor() throws UndefinedColorException {
        return 0xFF;
```

```
    }

    public static void main(String[] args) {
        ColoredStuff colored = new Pony();
        System.out.println( colored.getColor() );
        if (colored instanceof RideableStuff) {
            RideableStuff rideable = (RideableStuff) colored;
            System.out.println( rideable.getSpeed() );
        }
    }
}
```

Feladat

Csináljatok egy Bárány osztályt, amelynek van egy név attribútuma! Az osztály implementálja a java.lang.Cloneable interfészt! A main() függvényben példányosítsátok az osztályt, a neve legyen Dolly, majd klónozzátok az Object#clone() függvénnyel! Ezután írjátok ki a klón nevét!

Megjegyzés A visszatérési értékre specializálhatjátok a clone() függvényt:

```
@Override
public Sheep clone() throws CloneNotSupportedException { ... }
```

A dobott kivételt kezeljétek le!

Jar fájlok

Java Archive, appletek aláírhatók (felemelt biztonsági korlát), tömörít (sima zip file), hordozhatóság (pl. mobilra), klikkre indul, verziókövetés (manifest entry-ken keresztül).

```
jar --help
```

Fontosabb műveletek:

jar tf foo.jar	Tartalom listázása
jar xf foo.jar	Kicsomagolás
java -jar foo.jar	Futtatás
jar cf mibe.jar miket	Új jar létrehozása

Új jar létrehozása

Alakja:

```
jar cf [foo.jar] [miből]
```

- c : Ez a kapcsoló azt mutatja, hogy szeretnénk létrehozni egy új archívumot. Konvenció, hogy jarnak nevezzük ezeket az állományokat, de ez nem kötelező.
- f : Ezzel mondjuk meg a programnak, hogy fájlból szeretnénk dolgozni. E nélkül az std. inputról várja a bemenetet.
- miből : Ezeket a fájlokat csomagoljuk be. Ezek lehetnek egész könyvtárak, könyvtárstruktúrák vagy csak 1 darab classfájl.

Miután létrehoztuk az archívumot, még van egy apró dolgunk: módoítanunk kell a manifest[1]-et, hogy megmondjuk mely osztályból kell indítani a main() függvényt. Erre csak akkor van szükség, ha futtathatóvá szeretnénk a tenni a jart. Ezt úgy tehetjük meg, hogy létrehozunk egy sima txt fájlt (pl. mainClass.txt), amibe egyetlenegy sort kell begépelni:

```
Main-Class: MAIN_CLASS_NEVE (pl. Hellow.class esetén Hellow)
```

Gotcha A fájl utolsó sora tartalmazzon egy üres sort, mert *az utolsó sor nem lesz feldolgozva!*

Ezután frissíteni kell a manifestet (update):

```
jar umf mainClass foo.jar
```

Futtatni vagy klikkeléssel, vagy a következő paranccsal lehet:

```
java -jar foo.jar
```

Feladat

Készíts egy futtatható jar fájlt a következő HelloWorld programból! A package struktúra megtartására figyelj!

```
package pkg;

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

[1] A manifest egy olyan fájl, ami automatikusan generálódik a létrehozásnál, de akár le is cserélhető. Az a lényege, hogy metainformációt tartalmaz az archívumról, pl. ki, mikor fordítota, milyen fordítóval, mi a futtatandó osztály, milyen **classpath beállítások** tartoznak hozzá (ha van benne ilyen, mást nem is vesz figyelembe), etc.

Streamek

Csatornák, absztrakciós szint bemenet-kimenet kezelésére (hiext, lorem).

Részletesen: `http://java.sun.com/docs/books/tutorial/essential/io/`

Példa

```
package gyak3;

import java.io.FileWriter;
import java.io.PrintWriter;

public class WriteSampleFile {
    public static void main(String[] args) throws Exception {
        PrintWriter pw = new PrintWriter( new FileWriter("dummy.txt") );
        pw.println("Dummy data here");
        pw.close();
    }
}
```

Csoportosítás

Szervezés szerint:

- InputStream, OutputStream (bájtszervezésű)
- Reader, Writer (karakterszervezésű)

Ezekből rengeteg változat, pl.

- StringReader, FileReader, BufferedReader, etc.
- FileInputStream, DataInputStream, etc.
- FileOutputStream, PrintStream, etc.
- FileWriter, PrintWriter, etc.

Pl. FileReader, FileWriter, FileInputStream, FileOutputStream
Feladat szerinti csoportosítás:

- Adatforrás, adatnyelő (pl. FileInputStream, FileReader)
- Szűrők: meglévő csatornához plusz funkcionalitás (pl. BufferedInputStream). Csatornák összekapcsolhatók (pl. SequenceInputStream), csatornák (pl. kommunikációs közeg szálaknak, PipedInputStream, etc.)

Alapvető műveletek

- Megnyitás automatikus, lezárás: `close()` (ne felejtsd el, mert nem fogsz tudni hozzáférni)
- flush(): automatikus endline karakterre a szöveges fájlokban.
- Kiírás: write(), print()
- Olvasás: read(). Ha a csatorna kiürül, akkor az olvasó művelet blokkolódik, amíg adatot nem kap:

```
public static void main(String[] args) throws Exception {
    int i = System.in.read();
    System.out.println("A kapott karakter: " + i);
}
```

- "Könyvjelző-mechanizmus", ha támogatott: markSupported(), mark(), reset()
- Padding átugrása: skip()
- Csatorna ürességének ellenőrzése: ready()
- Hány bájtot lehet minimum olvasni? available()

```
int meret = new FileInputStream("tmp.txt").available();
```

Speciális streamek

1. Sorok számolására: LineNumberInputStream#getLineNumber()
2. Adatok olvasása, visszafűzése a csatornára: PushbackInputStream, PushbackReader
3. Bufferelt csatornák: BufferedReader, BufferedInputStream
4. Véletlen elérésű fájlok: RandomAccessFile
5. Szövegfeldolgozás: StringTokenizer (van sima StreamTokenizer is)
6. System.in, out, err: ezek is InputStream, PrintStream-ek. Pl. std. input átirányítása:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Fájl írása

```
package gyak3;

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 gyak3;

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) {
        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>_gyak<gyakorlat száma>_<EHA-kód>_<a megoldott feladatok száma>
```

Például:

```
csop1_gyak3_LERIAAT_1  (csak az első feladattal)
csop1_gyak3_LERIAAT_12 (mindkét feladattal)
```

Mellékelni csak a Java forrásfájlokat mellékeljétek (semmiképp ne teljes Eclipse/NetBeans projecteket), esetleg rarolva, zipelve, 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

Készítsünk egy egyszerű l33t5p34k (leetspeak) generátort! A program 1 parancssori argumentumot kapjon: egy input fájl elérési utat. A program olvassa be az inputot fájlt, és minden szón végezze el a következő módosításokat, majd írja ki a képernyőre a módosított szöveget:

- Minden 2. karakter esetén a kisbetűből csináljon nagyot, a nagybetűből kicsit!

- "a" → "@"
- "e" → "3"
- "i" → "1"
- "o" → "0"
- "u" → "v"
- "f" → "ph"
- "s" → "$"
- "g" → "9"
- "y" → "j"
- "t" → "+"
- "!" → "1"
- Ha az utolsó karakter:
  - "s" → "z"
  - "ck" → "x"

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!

Példa input fájl sorok:

```
123456789123456789123456789123456789123456789123456789123456789123456789
123456789234567891345678912456789123567891234678912345891234567891234567912345678
```