

Jegyzőkönyv

Szilágyi-Czumbil Ede Balázs

Jegyzőkönyv

Szilágyi-Czumbil Ede Balázs

Table of Contents

I.	1
1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroway!”	5
OO szemlélet	5
„Gagyí”:	6
Yoda	8
4. 2. hét - „Helló, Liskov!”	11
Liskov helyettesítés sértése	11

Part I.

Table of Contents

1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroway!”	5
OO szemlélet	5
„Gagyí”:	6
Yoda	8
4. 2. hét - „Helló, Liskov!”	11
Liskov helyettesítés sértése	11

Chapter 1. Infók

Neptun kód: CMY9W3

Git repó: <https://github.com/edeszilagy/Prog2>

e-mail: <ede.szilagyi@yahoo.com>

Chapter 2. 0. hét - „Helló, Berners-Lee!

Java, C++ összehasonlítás

Könyvek: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II

A Java nyelv teljesen objektum orientált nyelv. Ezzel szemben a C++ lehetőséget ad a generikus programozásra is. A két programnyelv változó kezelése eltér. Java esetén minden objektum referencia. Ez azt jelenti, hogy az értékeket közvetlen a referencián keresztül érjük el. Mindkét nyelv támogatja a publikus, privát és statikus objektum kezelést. Fordító szempontjából amíg a C++ kódot elég natívan fordítani, addig a Java-hoz szükség van egy virtuális fordítóra, ami futtatja a kódot. Emiatt nagyobb az erőforrás igénye is. Java-ban nem nagyon kell foglalkozni a memória szeméttel, mivel van automatikus garbage collector, ami üríti azt, míg C++-nál fel kell szabadítani a memóriát

Python

Könyv: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven

A Python tulajdonképpen egy szkriptnyelv, de nagyon sok csomagot is és beépített eljárást is tartalmaz, ezért komolyabb alkalmazások megírására és komolyabb problémák megoldására is használható. Más modulokkal is együtt tudunk kódolni egy Python komponens. A Python egy nagyon magas szintű programozási nyelv. Python esetén nincs szükség fordítás-ra. A Python interpreter elérhető számos platformon. A Pythont könnyű használni, megbízható és jelentős támogatást biztosít hibák javítására. A Pythonban minden adat objektumként szerepel. A rajtuk végzendő műveleteket az objektum típusa határozza meg, amit a rendszer futási időben határoz meg, így nekünk nem kell megadni. A következő típusok lehetnek: szám, string, tuple, list, dictionary. A számok lehetnek egészek, decimálisak, oktálisak vagy akár hexadecimálisak is. Szöveg típus esetén a szöveget két aposztróf közé írva kell megadni.

Chapter 3. 1. hét - „Helló, Arroway!”

OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, # amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algorit.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! Segédlink: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf (16-22 fólia)

A kód:

```
public class PolarGenerator {

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator() {
        nincsTarolt = true;
    }

    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();

                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;

                w = v1 * v1 + v2 * v2;
            } while(w > 1);
            double r = Math.sqrt(-2 * Math.log(w) / w);

            tarolt = r * v2;

            nincsTarolt = !nincsTarolt;
            return r * v1;
        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

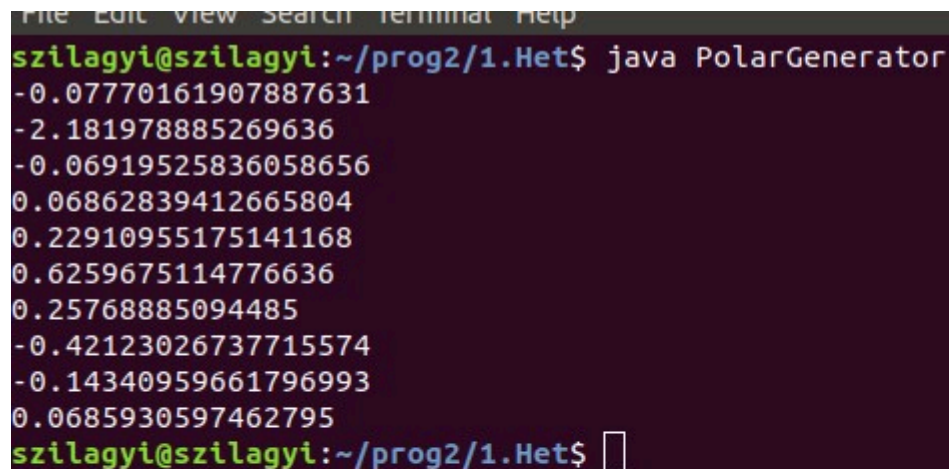
    public static void main(String[] args) {
        PolarGenerator pg = new PolarGenerator();

        for(int i = 0; i < 10; i++) {
            System.out.println(pg.kovetkezo());
        }
    }
}
```

A program kezdésként két változót állít el#. Magát a tárolt számot(double) és egy boolean típusú változót ami tárolja hogy van-e változó. Ezután ellen#rzi hogy van-e tárolt változó, ha van akkor generál

2 random számot amivel elvégzi az adott m#veletet. Ezt addig folytatja amíg a kapott eredmény kisebb lesz 1nél . Ha a nincs tárolt változó false akkor visszaadja a tárolt változóban lévő értéket.

Miután futtatuk:



```
File Edit View Search Terminal Help
szilagyi@szilagyi:~/prog2/1.Het$ java PolarGenerator
-0.07770161907887631
-2.181978885269636
-0.06919525836058656
0.06862839412665804
0.22910955175141168
0.6259675114776636
0.25768885094485
-0.42123026737715574
-0.14340959661796993
0.0685930597462795
szilagyi@szilagyi:~/prog2/1.Het$
```

„Gagy”:

Az ismert formális2 „while (x <= t && x >= t && t != x);” tesztkérdéstípusra adj a szokásosnál (miserint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írd Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

A JDK forrásán belül, a java/lang/Integer.java forrásban látszik hogy az Integer class-nak van egy alapértelmezett cache-je amiben vannak előre elkészített integer osztálybeli objektumok itt el vannak tárolva a -128-tól 127-ig terjedő számok hogy segítse a programok gyorsabb működését és jobb memóriahasználatát.

```
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

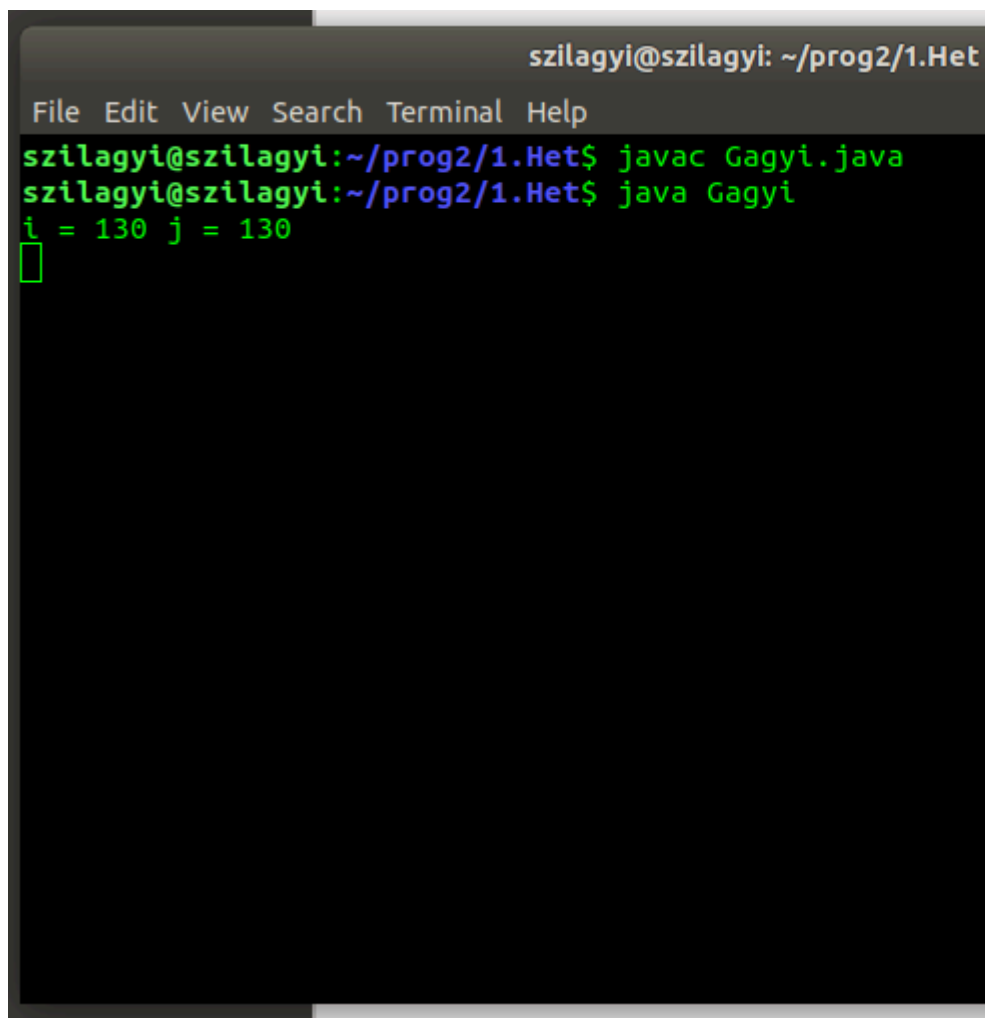
E miatt ha a programunkban a [-128,127] intervallumon kívüli értéket adunk meg akkor az egyenlőség hamis lesz mivel két új objektum fog létrejönni és emiatt végtelen ciklust kapunk.

Ha viszont az értékek az intervallumon belül vannak akkor igaz lesz az egyenlőség.

a program ami végtelen ciklust ad:

```
public class Gagyí {  
  
    public static void main(String[] args) {  
  
        Integer i = 130;  
        Integer j = 130;  
  
        System.out.println("i = " + i + " j = " + j);  
  
        while(i <= j && i >= j && i != j) {  
  
        }  
  
    }  
}
```

lefuttatva:



```
szilagyi@szilagyi: ~/prog2/1.Het  
File Edit View Search Terminal Help  
szilagyi@szilagyi:~/prog2/1.Het$ javac Gagyí.java  
szilagyi@szilagyi:~/prog2/1.Het$ java Gagyí  
i = 130 j = 130  
█
```

a program ami nem végtelen ciklust ad:

```
public class Gagyí {  
  
    public static void main(String[] args) {
```

```
Integer i = 10;
Integer j = 10;

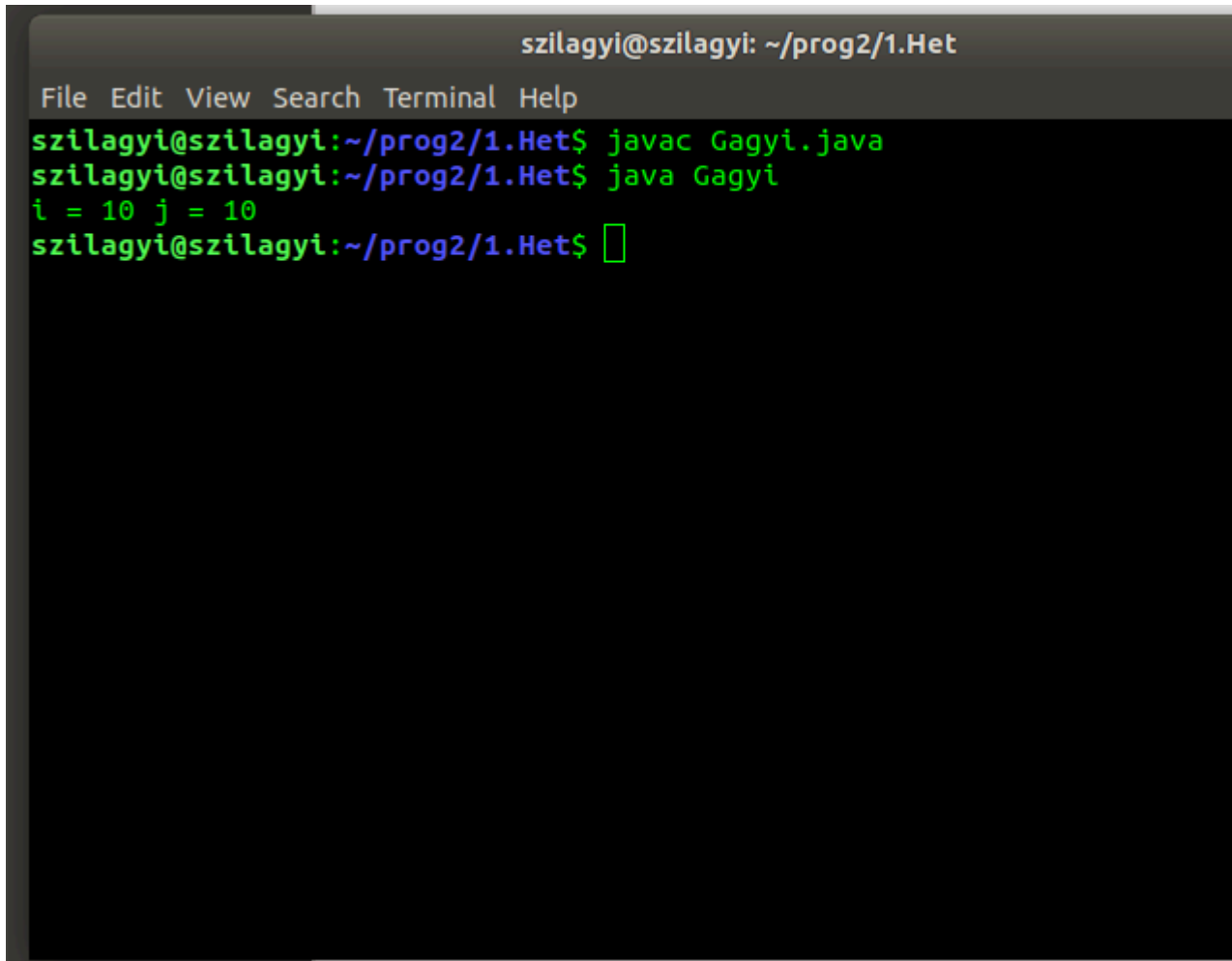
System.out.println("i = " + i + " j = " + j);

while(i <= j && i >= j && i != j) {

}

}
```

lefuttatva:



```
szilagyi@szilagyi: ~/prog2/1.Het
File Edit View Search Terminal Help
szilagyi@szilagyi:~/prog2/1.Het$ javac Gagy1.java
szilagyi@szilagyi:~/prog2/1.Het$ java Gagy1
i = 10 j = 10
szilagyi@szilagyi:~/prog2/1.Het$
```

Yoda

Írjunk olyan Java programot, ami `java.lang.NullPointerException`-el leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

A kód:

```
public class Yoda {

    public static void main(String[] args) {
```

```
final String str = null;

try {
    if(str.equals("...")) {
        //Do something
    }
    System.out.println("1. Success");

} catch(Exception e) {
    System.err.println(e.getMessage());
}

try {
    if("..." equals(str)) {
        //Do something
    }
    System.out.println("2. Success");

} catch(Exception e) {
    System.err.println(e.getMessage());
}
}
```

A Yoda condition olyan programozási stílus ahol a kifejezések fordított sorrendben vannak a tipikus, megszokott sorrendhez képest.

Az első programrészlet pont ezt próbálja szemléltetni. amikor először próbáljuk ellenőrizni az egyenlőséget NullPointerException-el leáll mivel megsértettük a Yoda condition-t (null), de mikor megcseréltük a sorrendet már sikerrel jártunk (Success)

Lefuttatva:

```
szilagyi@szilagyi: ~/prog2/1.Het
File Edit View Search Terminal Help
szilagyi@szilagyi:~/prog2/1.Het$ java Yoda
null
2. Success
szilagyi@szilagyi:~/prog2/1.Het$
```

Chapter 4. 2. hét - „Helló, Liskov!”

Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

C++ kódcsipet amely megsérti a Liskov elvet:

```
// ez a T az LSP-ben
class Madar {
public:
    virtual void repul() {};
};

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};

// itt jönnek az LSP-s S osztályok
class Sas : public Madar
{};

class Pingvin : public Madar // ezt úgy is lehet/kell olvasni, hogy a pingvin t
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin ); // sérül az LSP, mert a P::fgv röptetné a Pingvin
}
```

Így lenne helyes:

```
// ez a T az LSP-ben
class Madar {
//public:
// void repul(){};
};

// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
    void fgv ( Madar &madar ) {
        // madar.repul(); a madár már nem tud repülni
    }
}
```

```
        // s hiába lesz a leszármazott típusoknak
        // repül metódusa, azt a Madar& madar-ra úgysem lehet hívni
    }
};

// itt jönnek az LSP-s S osztályok
class RepuloMadar : public Madar {
public:
    virtual void repul() {};
};

class Sas : public RepuloMadar
{};

class Pingvin : public Madar // ezt úgy is lehet/kell olvasni, hogy a pingvin t
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin );
}
```