

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

Copyright © 2019 Bakos Bálint

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i>	
	Univerzális programozás	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Bakos, Bálint	2019. november 27.

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Második felvonás	3
2. Helló, Berners-Lee!	5
2.1. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihámér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.II.	5
2.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus fejlesztés Python és Java nyelven	6
3. Helló, Arroway!	7
3.1. OO szemlélet	7
3.2. "Gagyi"	14
3.3. Yoda	15
3.4. Kódolás from scratch	16
4. Helló, Liskov!	22
4.1. Liskov helyettesítés sértése	22
4.2. Szülő-gyerek	24
4.3. Anti OO	26
4.4. Ciklomantikus komplexitás	27

5. Helló, Mandelbrot!	29
5.1. Reverse engineering UML osztálydiagram	29
5.2. Forward engineering UML osztálydiagram	30
5.3. BPMN	32
5.4. BPEL Helló, Világ!	32
6. Helló, Chomsky!	33
6.1. Encoding	33
6.2. I334d1c4^5	35
6.3. Full screen	38
6.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció	42
7. Helló, Stroustrup!	44
7.1. JDK osztályok	44
7.2. Másoló-mozgató szemantika és Összefoglaló	46
7.3. Változó argumentumszámú ctor	53
8. Helló, Gödel!	57
8.1. Gengszterek	57
8.2. STL map érték szerinti rendezése	57
8.3. Alternatív Tabella rendezése	60
8.4. GIMP Scheme hack	61
9. Helló, Valaki!	63
9.1. FUTURE tevékenység editor	63
9.2. SamuCam	65
9.3. BrainB	68
10. Helló, Lauda!	72
10.1. Port scan	72
10.2. Android Játék	73
10.3. Junit teszt	82

11. Helló, Calvin!	85
11.1. MNIST	85
11.2. Deep MNIST	85
11.3. CIFAR-10	85
11.4. Android telefonra a TF objektum detektálója	85
11.5. SMNIST for Machines	90
11.6. Minecraft MALMO-s példa	90
 III. Irodalomjegyzék	 94
11.7. Általános	95
11.8. C	95
11.9. C++	95
11.10Lisp	95

Táblázatok jegyzéke

4.1. Összehasonlítás	26
--------------------------------	----

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [[METAMATH](#)]

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz alkalmi igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Minden esetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyereknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyereknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk más is) példával.

Hogyan nyomjuk?

Rántsd le a <https://gitlab.com/nbatfai/bhax> git repót, vagy méginkább forkolj belőle magadnak egy sajátot a GitLabon, ha már saját könyvön dolgozol!

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dblatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml ←
    --noout
output.xml validates
rm -f output.xml
dblatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

Ha minden igaz, akkor most éppen ezt a legenerált **bhax-textbook-fdl.pdf** fájlt olvasod.



A DocBook XML 5.1 új neked?

Ez esetben forgasd a <https://tdg.docbook.org/tdg/5.1/> könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

Mi a programozás?

Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.

II. rész

Második felvonás

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

2. fejezet

Helló, Berners-Lee!

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

A C++ valamint a Java nyelv is magasszintű programozási nyelvnek számít. Mindkettő objektumorientált, azonban mivel a C++ nyelv hamarabb alakult ki így kisebb-nagyobb eltérések találhatóak a két nyelv között.

A C++ a C nyelvtől örökolt gépközeli konstrukciókat, ebből adódik sebessége. A Java nyelv pedig a C++-ból vett át sok minden. Azonban van egy lényeges eltérés, hogy az Java nyelv a mutatók helyett referenciakat használ, így biztonságosabb, megbízhatóbb programokat lehet írni. Valamint a prgoramok hordozhatósága is eltér. Például ha egy Linuxon írt C++ kódot akarunk átvinni Windowsra az nem biztos, hogy működni fog, de ha egy Java kódot akarunk futtani máshol az jól fog szuperálni feltéve, hogy van JVM(Java Virtual Machine) a gépen. Mivel ez a Java bájtkódot fogja futtani, így ez platformfüggetlen lesz.

Szintaktikában is találunk eltéréseket a két nyelv között.

Hasonlítsuk össze mondjuk ezt a két Hello world! programot.

C++-ban:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Java-ban:

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

```
        System.out.println("Hello World!");
    }
}
```

Amint látható a Java nagyrészben osztályalapú. Az osztályoknak különböző elérése lehet: public, protected, private. Az osztályokon belül létrehozhatunk változókat, függvényeket.

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

A Python támogatja a funkcionális valamint az imperatív nyelveket is. Legfőbb jellemzője, ami teljesen eltér a többi magasszintű programozási nyelvtől, hogy behúzásalapú a szintaxisa, tehát semmilyen kapcsos zárójelre vagy explicit kulcsszóra nincs szükség. Valamint a sorok végén már megszokott pontosvessző sem kell.

```
if 1 < 2
    print("Nagyobb!")
else:
    print("Kisebb!")
```

A Python egy objektumorientált nyelv, tehát minden adatot objektumok reprezentálnak. A változók típusainak explicit megadására sincs szükség, a rendszer futási időben dönti el a változók típusát.

Ilyen egyszerűen néz ki Phytonban egy dekrelálás:

```
name = "Ádám"
```

A nyelvben definiálhatunk osztályokat is és ezeknek példányai az objektumok. Az osztályok attribútumai lehetnek objektumok vagy függvények is.

3. fejezet

Helló, Arroway!

OO szemlélet

A polártranszformációs generátor egy széles körben elterjedt random generátor. Olyannyira elterjedt formája ez a random szám generálásnak, hogy a Java.util.Random osztály is ezt a módszert alkalmazza.

Íme a Java kód teljes egészében:

```
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 g = new PolarGenerator();

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

A következő sorokban részenként magyarázom el mi történik a kódban.

```
boolean nincsTarolt = true;
double tarolt;
```

A PolárGenerátor classban létrehozunk két változót. Az egyik boolean típusú, amely azt fogja megmondani, hogy éppen van-e tárolt értékünk. A másik maga a tárolt értéket tartalmazza, amit korábban kiszámítottunk.

```
public PolarGenerator() {
    nincsTarolt = true;
}
```

Ez a class publikus konstruktora, amely a nincsTarolt-at igazra állítja. Erre azért van szükség, hogy tudjuk éppen van-e tárolt érték vagy nincs.

```
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;
    }
}
```

Ez a program lelke ahol a kovetkezo() metódus végzi a meghatározó számítást. Ha a nincsTarolt értéke igaz, akkor számol két random értéket. Az egyiket elmenti a tarolt változóba, a másikat pedig visszaadja. Amennyiben a nincsTarolt értéke hamis, akkor pedig a tárolt értéket fogja visszaadni.

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

    PolarGenerator g = new PolarGenerator();

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

A main metódusal indul el a program. Itt példányosítjuk a PolárGenerátor classot. Ezután egy for ciklus-sal 10-szer kiíratjuk a meghívott kovetkezo() metódus értékét. minden második érték a tarolt változóból visszadott érték lesz. Ezeket az értékeket generálta nekem:

```
bunyi@bunyi:~/Documents$ javac PolarGenerator.java
bunyi@bunyi:~/Documents$ java PolarGenerator
-0.5518103598184344
-0.9108738027998466
-1.9560282490949241
0.5048225085719978
1.9118928849455419
-0.48283343220250585
1.7238209676208334
0.8937594137943287
-1.3956361879015262
0.8904870819477649
bunyi@bunyi:~/Documents$
```

Az érdekesség az még itt, hogy a Java fejlesztők egy nagyon hasonló módon oldották meg a java.util.Random osztályban a Random szám generálást. Íme:

```
public double nextDouble() {
    return (((long)(next(26)) << 27) + next(27)) * DOUBLE_UNIT;
}

private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}
```

C++-ban így néz ki a teljes kód:

```
#include <iostream>
#include <tgmath.h>
#include <cstdlib>
#include <time.h>

using namespace std;

class PolarGenerator {
private:
    bool nincsTarolt;
    double tarolt;

public:
    PolarGenerator() {
        nincsTarolt = true;
        srand (time(NULL));
    }

    double kovetkezo() {
        if (nincsTarolt) {
```

```
double u1, u2, v1, v2, w;

do {
    u1 = rand() / (RAND_MAX + 1.0);
    u2 = rand() / (RAND_MAX + 1.0);

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

    w = v1 * v1 + v2 * v2;
} while (w > 1);

double r = sqrt((-2 * log(w)) / w);
tarolt = r * v2;
nincsTarolt = !nincsTarolt;

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

};

int main(int argc, char** argv) {
    PolarGenerator g;

    for (int i = 0; i < 10; ++i)
        cout << g.kovetkezo() << endl;

    return 0;
}
```

A következő sorokban részenként magyarázom el mi történik a kódban.

```
private:
    bool nincsTarolt;
    double tarolt;
```

Ez a Polárgenerátor class private része. Ez tartalmaz egy bool és egy double típusú változót. Ezek a változók csak az osztályon belül lesznek elérhetőek.

```
public:
    PolarGenerator() {
        nincsTarolt = true;
        srand (time(NULL));
    }
```

```
double kovetkezo() {
    if (nincsTarolt) {
        double u1, u2, v1, v2, w;

        do {
            u1 = rand() / (RAND_MAX + 1.0);
            u2 = rand() / (RAND_MAX + 1.0);

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

            w = v1 * v1 + v2 * v2;
        } while (w > 1);

        double r = sqrt((-2 * log(w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

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

Ez pedig a class public része, amelyben a változók és metódusok példányosítás után elérhetőek az osztályon kívül is.

```
PolarGenerator() {
    nincsTarolt = true;
    srand (time(NULL));
}
```

Az osztály nevével megegyező metódust konstruktornak nevezzük. Az ebben lévő kódok példányosításkor hajtódnak végre.

```
double kovetkezo() {
    if (nincsTarolt) {
        double u1, u2, v1, v2, w;

        do {
            u1 = rand() / (RAND_MAX + 1.0);
            u2 = rand() / (RAND_MAX + 1.0);

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

            w = v1 * v1 + v2 * v2;
    } while (w > 1);
```

```
        double r = sqrt((-2 * log(w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

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

A következő metódusban szinte semmilyen lényegi eltérés nincs a Java kódhoz képest. Ugyanaz történik ha nincsTarolt igaz akkor generál randomot, ha pedig hamis, akkor visszaadja az eltárolt értéket.

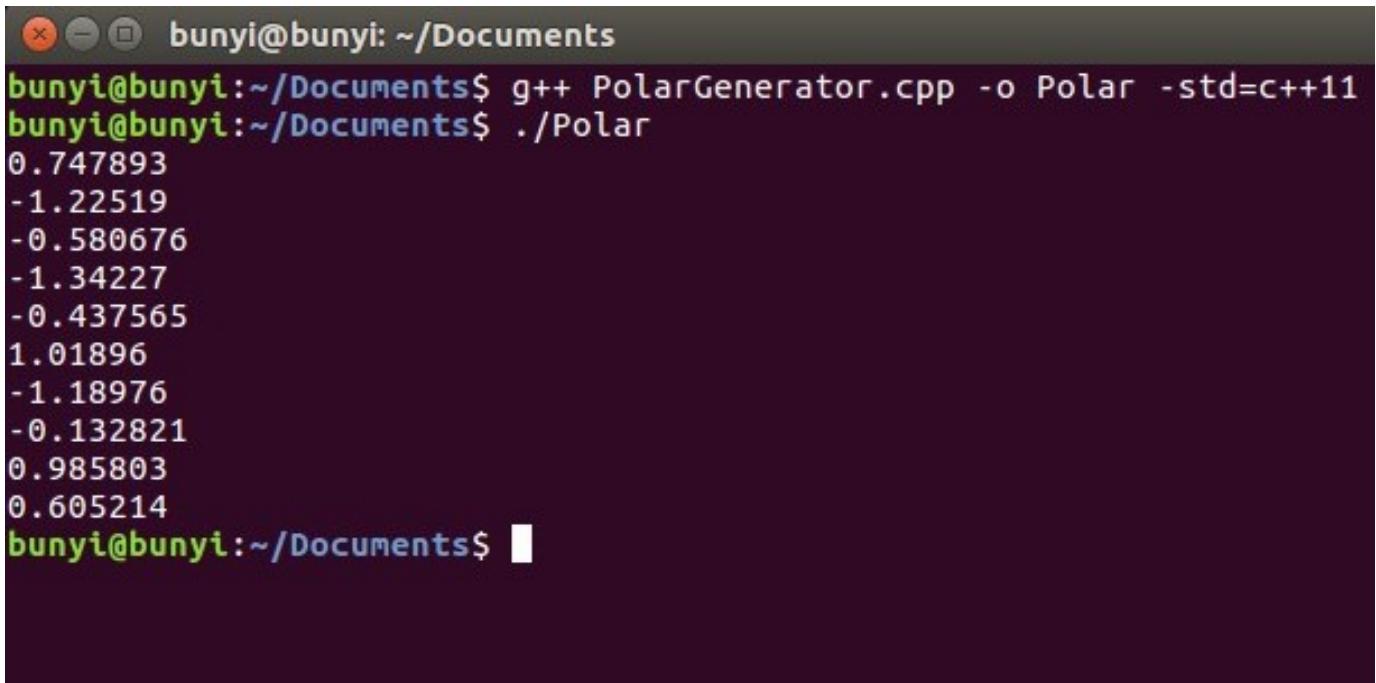
```
int main(int argc, char** argv) {
    PolarGenerator g;

    for (int i = 0; i < 10; ++i)
        cout << g.kovetkezo() << endl;

    return 0;
}
```

A main metódusban szintén, mint a Javanál példányosítunk és egy for ciklussal 10-szer visszadjuk a kovetkezo() metódus értékét.

C++-ban generált értékek:



```
bunyi@bunyi:~/Documents$ g++ PolarGenerator.cpp -o Polar -std=c++11
bunyi@bunyi:~/Documents$ ./Polar
0.747893
-1.22519
-0.580676
-1.34227
-0.437565
1.01896
-1.18976
-0.132821
0.985803
0.605214
bunyi@bunyi:~/Documents$
```

"Gagyi"

```
while (x <=t && x>=t && t !=x);
```

Erre a tesztkérdésre kellett választ adnunk, hogy bizonyos számoknál miért jön létre végtelen ciklus és bizonyosnál miért nem.

```
public class Gagyi {  
  
    public static void main (String[]args){  
  
        Integer x = -128;  
        Integer t = -128;  
  
        while (x <= t && x >= t && t != x);  
    }  
}
```

Például itt -128-nál nem jön létre végtelen ciklus.

```
public class GagyiInfinity {  
  
    public static void main (String[]args){  
  
        Integer x = -129;  
        Integer t = -129;  
  
        while (x <= t && x >= t && t != x);  
    }  
}
```

Azonban -129-el már végtelen ciklus jön létre.

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

A válasz, hogy miért jön létre -129-el végtelen ciklus, míg -128-al semmi sem történik ebben a kódcsipetben rejlik, ami a java.lang.Integer osztályban található.

Elsősorban mindenki által tudni kell, hogy a !=, == operátorok az objektumok címét hasonlítják össze, valamint a Java feltételezi, hogy a programok sokat dolgoznak, majd kis számokkal, így a poolban már előre elkészített számok vannak 127-től -128-ig. Tehát amikor létrehozunk két Integer objektumot és az a poolon belül van, akkor a két objektum címe meg fog egyezni. Pontosan ez történik a -128-nál, lérehozzuk a két objektumot x-et és y-ot, azonban a poolból kapjuk meg mindkettőt egy már előre elkészített objektumot így a cím megegyezik. Ezért a x != y hamis értéket fog adni, így a while ciklus feltétele nem teljesül és nem jön létre végtelen ciklus.

```
return new Integer(i);
```

Ha nem esik bele viszont a poolba a szám akkor, itt látszik, hogy létrehoz egy új Integert. Mivel a -129 nem esik bele így két különböző című objektumot fog létrehozni és így a while ciklus feltétele igaz lesz és végtelen ciklust kapunk.

Yoda

A feladat az volt, hogy írunk egy olyan Java kódot ami NullPointerException hibával kilép, ha nem követjük a Yoda conditionst. Íme a kód:

```
public class Yoda {  
  
    public static void main(String[] args) {  
  
        String myString = null;  
  
        if ("something".equals(myString)) {  
            System.out.println("True");  
        } else {  
            System.out.println("False");  
        }  
  
        //NullPointerException  
        if (myString.equals("something")) {  
            System.out.println("True");  
        } else {  
            System.out.println("False");  
        }  
    }  
}
```

A Yoda conditions egy kódolási stílus, ahol a programkódot "fordítva" írjuk be, tehát az értékkedásnál a konstans értéket írjuk balra és jobbra kerül a változó amibe elmentjük. A nevét is erről a szokatlan megfordított kódírásról kapta, Yoda-ról aki a Star Wars-ban hasonlóan nem szabályszerűen alkalmazza az angolt.

```
int érték = 3;  
if( érték == 3) {  
    System.out.println("Igaz");  
}
```

Ez ahogyan rendesen írnánk egy kódot.

```
int érték = 3;  
if( 3 == érték) {  
    System.out.println("Igaz");  
}
```

Ugyanaz a kód Yoda conditions-t használva. Mindakettővel teljesen normálisan fog működni a program.

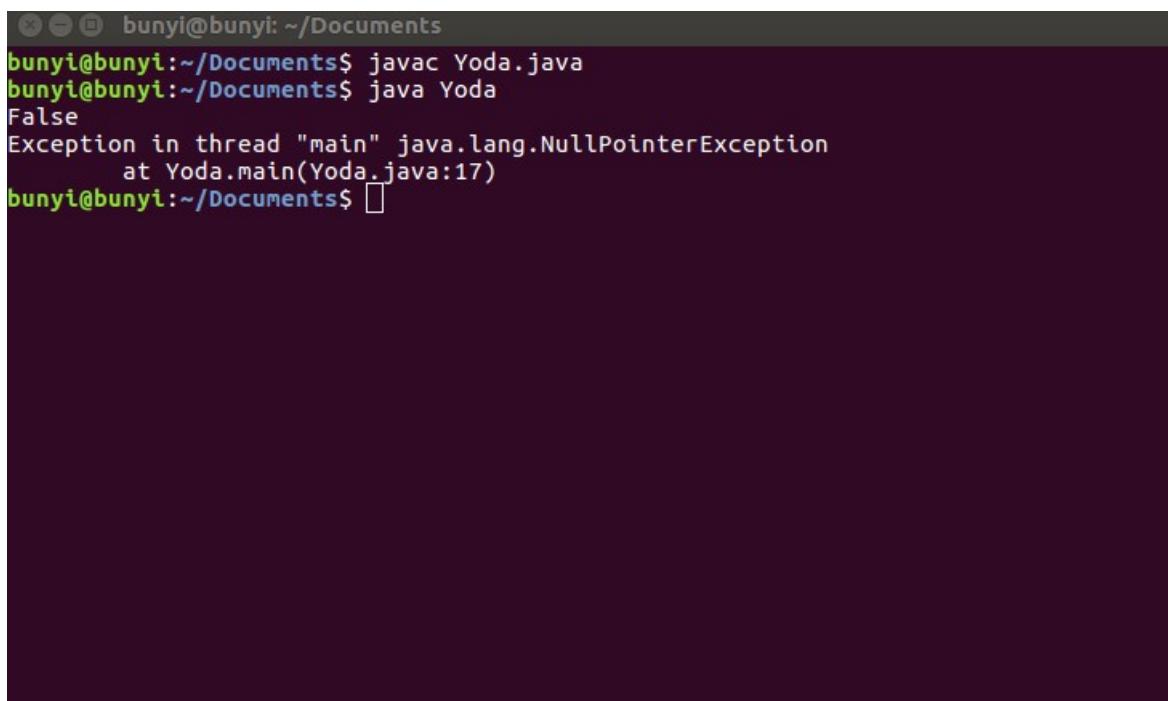
```
if ("something".equals(myString)) {  
    System.out.println("True");
```

```
} else {
    System.out.println("False");
}
```

Ez a rész egy tipiukan Yoda conditions-t használva lett megírva. Látszik, hogy az equal metódus bal oldalára került a konstans, jelen esetben egy string. Az equal ezt hasonlítja össze a myStringben lévő null értékkel. Ekkor semmilyen hiba nem fordul elő egyszerűen hamis lesz a visszadott érték.

```
if (myString.equals("something")) {
    System.out.println("True");
} else {
    System.out.println("False");
}
```

Ezek a sorok azonban nem használják a Yoda conditions-t, így NullPointerException-t dobnak a Java-ban. Így a Yoda conditions-al elkerülhető néhány nem biztonságos null viselkedés.



```
bunyi@bunyi:~/Documents$ javac Yoda.java
bunyi@bunyi:~/Documents$ java Yoda
False
Exception in thread "main" java.lang.NullPointerException
        at Yoda.main(Yoda.java:17)
bunyi@bunyi:~/Documents$ ]
```

Amint látszik a console-on is NullPointerException hibával kilép.

Azonban a Yoda conditions bírálói nagyban panaszkodnak az olvashatóság elvesztésére, ha ezt a módszert alkalmazzuk.

Kódolás from scratch

Egy olyan feladatot kaptunk, hogy írjuk meg a BBP algoritmus megvalósítását. Ez egy olyan algoritmus ami kiszámítja a Pi hexadecimális számjegyeit egy megadott helyen. Íme a kód:

```
public class BBP {
    String HexaJegyek;
```

```
public BBP(int d) {  
  
    double HexPi = 0.0;  
  
    double S1 = Sj(d, 1);  
    double S4 = Sj(d, 4);  
    double S5 = Sj(d, 5);  
    double S6 = Sj(d, 6);  
  
    HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;  
  
    HexPi = HexPi - Math.floor(HexPi);  
  
    StringBuffer sb = new StringBuffer();  
  
    Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};  
  
    while(HexPi != 0.0) {  
  
        int jegy = (int)Math.floor(16.0d*HexPi);  
  
        if(jegy<10) {  
            sb.append(jegy);  
        } else {  
            sb.append(hexaJegyek[jegy-10]);  
        }  
  
        HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);  
  
    }  
  
    HexaJegyek = sb.toString();  
}  
  
public String toString() {  
    return HexaJegyek;  
}  
  
public double Sj(int d, int j) {  
  
    double Sj = 0.0;  
  
    for (int k = 0; k <= d; k++)  
        Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    return Sj - Math.floor(Sj);  
}  
  
public long n16modk(int n, int k) {
```

```
int t = 1;
while(t <=n)
    t *= 2;

long r = 1;

while(true) {

    if(n >= t) {
        r = (16*r) % k;
        n = n - t;
    }

    t = t/2;

    if(t < 1)
        break;

    r = (r*r) % k;
}

return r;
}

public static void main(String[] args) {
    System.out.println(new BBP(1000000));
}
}
```

Nézzük részekre bontva:

```
String HexaJegyek;
```

Először is létrehozunk a BBP classban egy változót. Ebben a változóban fogjuk tárolni a végeredményt, tehát a Pi hexadecimális jegyeit az adott helyen.

Nézzük először a metódusokat, mert csak azután lehet megérteni a konstruktur működését.

```
public String toString() {
    return HexaJegyek;
}
```

Legegyszerűbb a a `toString()` metódussal kezdeni. Ez egyszerűen visszaadja a végső eredményt egy string-ként, de ebben az esetben az eredményünk eleve string típusú így nincs más dolgunk csak azt visszaadni.

```
public long n16modk(int n, int k) {

    int t = 1;
    while(t <=n)
        t *= 2;

    long r = 1;
```

```
while(true) {  
  
    if(n >= t) {  
        r = (16*r) % k;  
        n = n - t;  
    }  
  
    t = t/2;  
  
    if(t < 1)  
        break;  
  
    r = (r*r) % k;  
}  
  
return r;  
}
```

Az n16modk metódusban számoljuk ki bináris hatványozással a $16^n \bmod k$ értékét.

```
public double Sj(int d, int j) {  
  
    double Sj = 0.0;  
  
    for (int k = 0; k <= d; k++)  
        Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    return Sj - Math.floor(Sj);  
}
```

Az Sj metódus egy double értékkel fog visszatérni. A BBP algoritmus képlet alapján fogja visszaadni ezt a számot.

```
public BBP(int d) {  
  
    double HexPi = 0.0;  
  
    double S1 = Sj(d, 1);  
    double S4 = Sj(d, 4);  
    double S5 = Sj(d, 5);  
    double S6 = Sj(d, 6);  
  
    HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;  
  
    HexPi = HexPi - Math.floor(HexPi);  
  
    StringBuffer sb = new StringBuffer();  
  
    Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

```
while(HexPi != 0.0) {  
  
    int jegy = (int)Math.floor(16.0d*HexPi);  
  
    if(jegy<10) {  
        sb.append(jegy);  
    } else {  
        sb.append(hexaJegyek[jegy-10]);  
    }  
  
    HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);  
  
}  
  
HexaJegyek = sb.toString();  
}
```

Ez a rész a konstruktora a BBP classnak, ez mindenéppen le fog futni amikor példányosítják a függvényt. Nézzük ezt is részekre bontva:

```
double HexPi = 0.0;  
  
double S1 = Sj(d, 1);  
double S4 = Sj(d, 4);  
double S5 = Sj(d, 5);  
double S6 = Sj(d, 6);  
  
HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;  
  
HexPi = HexPi - Math.floor(HexPi);
```

Először is létrehoz 5 db változót. A HexPi-t azért, hogy legyen miben tárolni a számot amit a képlet kiszámolása után megkapunk. Az S1, S4, S5, S6 változók részelemek az alatta lévő képletben. A d változó az a szám itt amit a felhasználó ad meg, hogy hányadik helyen számolja a Pi hexadecimális értékét.

```
StringBuffer sb = new StringBuffer();  
  
Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};  
  
while(HexPi != 0.0) {  
  
    int jegy = (int)Math.floor(16.0d*HexPi);  
  
    if(jegy<10) {  
        sb.append(jegy);  
    } else {  
        sb.append(hexaJegyek[jegy-10]);  
    }  
  
    HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);
```

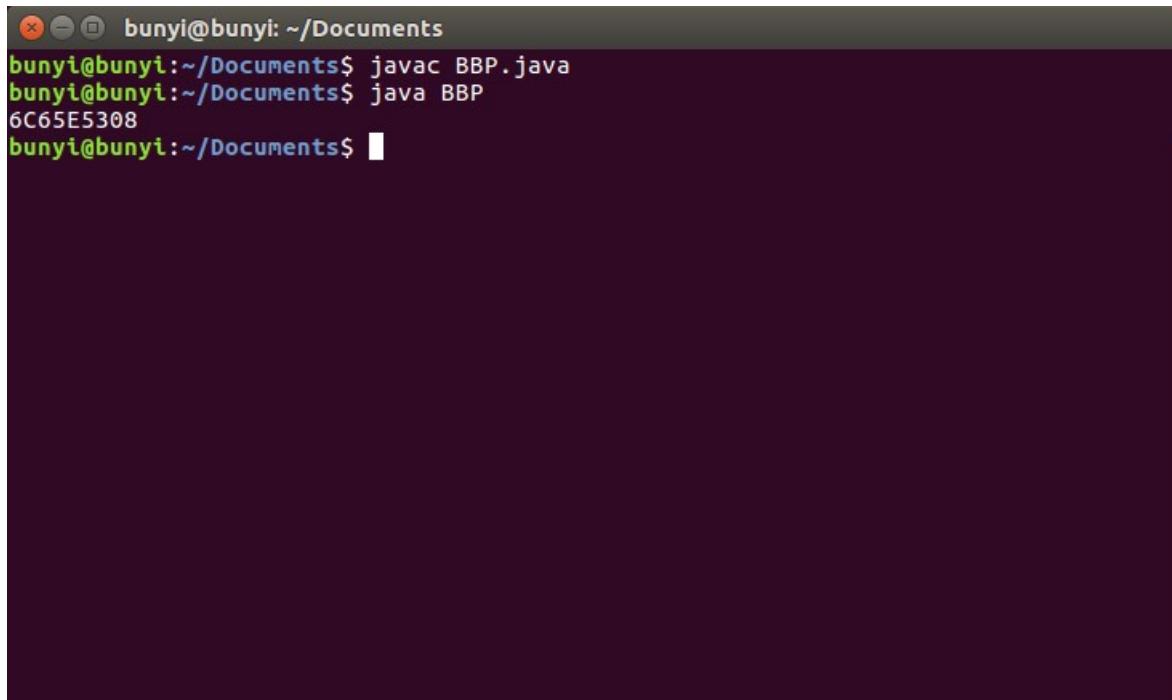
```
}
```

```
HexaJegyek = sb.toString();
```

Itt létrehozunk egy StringBuffert amiben ideiglenesen elmentjük a hexadecimális számokat stringként. Egy Character típusú tömböt is alkotunk, ebben tároljuk a 16-os számrendszerben jelenlévő karaktereket. A while cikluson belül a stringbuffer-hez hozzáfűzzük a hexa számjegyeket, majd a legalján átadjuk a HexaJegyek nevű stringnek.

```
public static void main(String[] args) {
    System.out.println(new BBP(1000000));
}
```

A main metódusban a kiíratáson belül egy példányosítást láthatunk 10^6 értékkel. Tehát a program a Pi 1 milliomodik helyen lévő hexadecimális számjegyeit fogja visszaadni. Ahogy látható is:



```
bunyi@bunyi:~/Documents$ javac BBP.java
bunyi@bunyi:~/Documents$ java BBP
6C65E5308
bunyi@bunyi:~/Documents$
```

4. fejezet

Helló, Liskov!

Liskov helyettesítés sértése

Ebben a feladatban egy objektum orientált kódot kellett írnunk Java és C++ nyelven, amely megséríti a Liskov elvet. Először is mi az a Liskov elv?

A liskov elvet Barbara Liskov mutatta be először. Fő célja a rossz OO tervezés megakadályozása. Az elv kimondja, hogy ha S altípusa T-nek, akkor bármely helyen ahol T-t alkalmazzuk S-t is minden probléma nélkül használhatjuk úgy, hogy a programrész tulajdonságai nem változnak.

Java kód:

```
static class Macska {
    public void szörös() { }
}

static class Program {
    void fgv (Macska macska) {
        macska.szörös();
    }
}

static class Perzsa extends Macska {}
static class Szfinx extends Macska {}

public static void main(String[] args) {
    Program program = new Program();
    Macska macska = new Macska();
    program.fgv(macska);

    Perzsa perzsa = new Perzsa();
    program.fgv(perzsa);

    Szfinx szfinx = new Szfinx();
    program.fgv(szfinx);
}
```

C++ kód:

```
class Macska {
public:
    virtual void szörös() {};
};

class Program {
public:
    void fgv(Macska& macska) {
        macska.szörös();
    }
};

class Perzsa : public Macska {};
class Szfinx : public Macska {};

int main(int argc, char** argv) {
    Program program;
    Macska macska;
    program.fgv(macska);

    Perzsa perzsa;
    program.fgv(perzsa);

    Szfinx szfinx;
    program.fgv(szfinx);
}
```

A kódokban a Macska nevű ősosztály a T, ennek gyermekosztályai pedig a Perzsa és a Szfinx ami itt az S. A Macska nevű osztály tartalmaz egy szörös() nevű függvényt, ez azt jelenti, hogy minden macska szörös, de ez nem igaz. A szfinx fajtájú macskák szőrtelenek. Azonban a program őt is szörösnek titulálja. A Liskov elv így sérül, mivel van olyan leszármazott, amely nem rendelkezik az őse tulajdonságával, így behelyettesíteni sem lehet az ős helyére a leszármazottat.

Úgy tudjuk kiküszöbölni ezt a hibát itt, hogy létrehozunk, még egy osztályt SzörösMacska néven amely a Macskából származik és ennek lesz leszármazottja a Perzsa.

```
static class Macska {}

static class Program {
    void fgv (Macska macska) {}
}

static class SzörösMacska extends Macska {
    public void szörös() {}
}

static class Perzsa extends SzörösMacska {}
static class Szfinx extends Macska {}
```

```
public static void main(String[] args) {
    Program program = new Program();
    Macska macska = new Macska();
    program.fgv(macska);

    Perzsa perzsa = new Perzsa();
    program.fgv(perzsa);

    Szfinx szfinx = new Szfinx();
    program.fgv(szfinx);
}
```

A fent látható megoldással már nem sérül a Liskov elv.

Szülő-gyerek

Ennek a feladatnak a megoldásához tisztázni kell az öröklődés és a polimorfizmus fogalmát. Az öröklődés az, amikor egy osztályt egy már létező osztály kiterjesztével definiálunk. Ekkor a már létező osztály lesz az ős- vagy szülőosztály. Az osztály amit pedig kiterjesztünk leszármazott vagy gyermekosztálynak nevezünk. A létrejött utód egy új osztály lesz, amely örökli az ős metódusait, tagváltozóit. A public, protected, private kulcsszavakkal lehet megadni, hogy a gyermek melyik metódusokat, változókat lássa. A public-al rendelkezőket mindenki használhatja, a protected-et csak az ős leszármazottai. A private tagot csak az az osztály, amelyben létrehozták a tagot.

A polimorfizmus lényege az, hogy mivel a gyermekosztály örökölt minden metódust és tagváltozót így egy olyan környezetben ahol az őst lehet használni a gyermeket is.

Példa arra, hogy az ősn keresztül csak az ős üzenetei küldhetőek. Javaban:

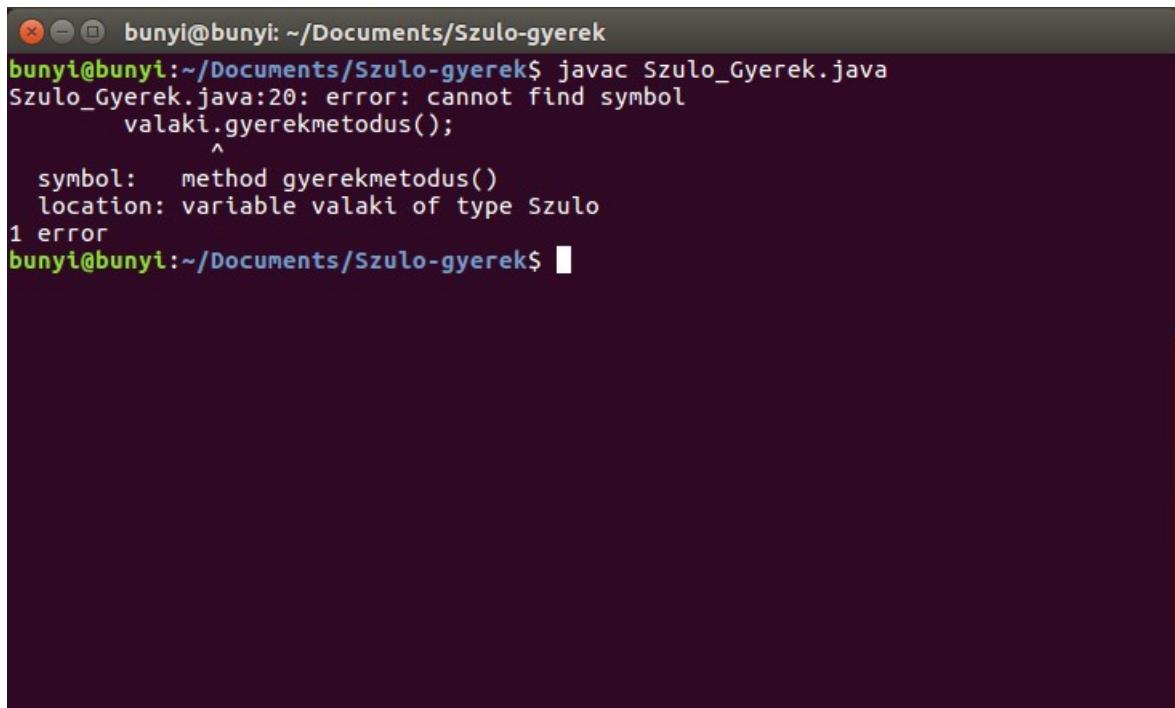
```
class Szulo {
    void szulometodus() {
        System.out.println("Szulo vagyok!");
    }
}

class Gyerek extends Szulo {
    void gyerekmetodus() {
        System.out.println("Gyerek vagyok!");
    }
}

class Szulo_Gyerek {
    public static void main(String[] args) {
        Szulo valaki = new Gyerek();

        valaki.szulometodus();
        valaki.gyerekmetodus(); //Nem látja a gyerekmetódusát!
    }
}
```

```
}
```



```
bunyi@bunyi:~/Documents/Szulo-gyerek
bunyi@bunyi:~/Documents/Szulo-gyerek$ javac Szulo_Gyerek.java
Szulo_Gyerek.java:20: error: cannot find symbol
    valaki.gyerekmetodus();
           ^
symbol:   method gyerekmetodus()
location: variable valaki of type Szulo
1 error
bunyi@bunyi:~/Documents/Szulo-gyerek$
```

Ahogy a console-on is látszik a program nem látja a gyermekmetódusát, azoban a gyermek látja a szülő metódusát.

C++-ban:

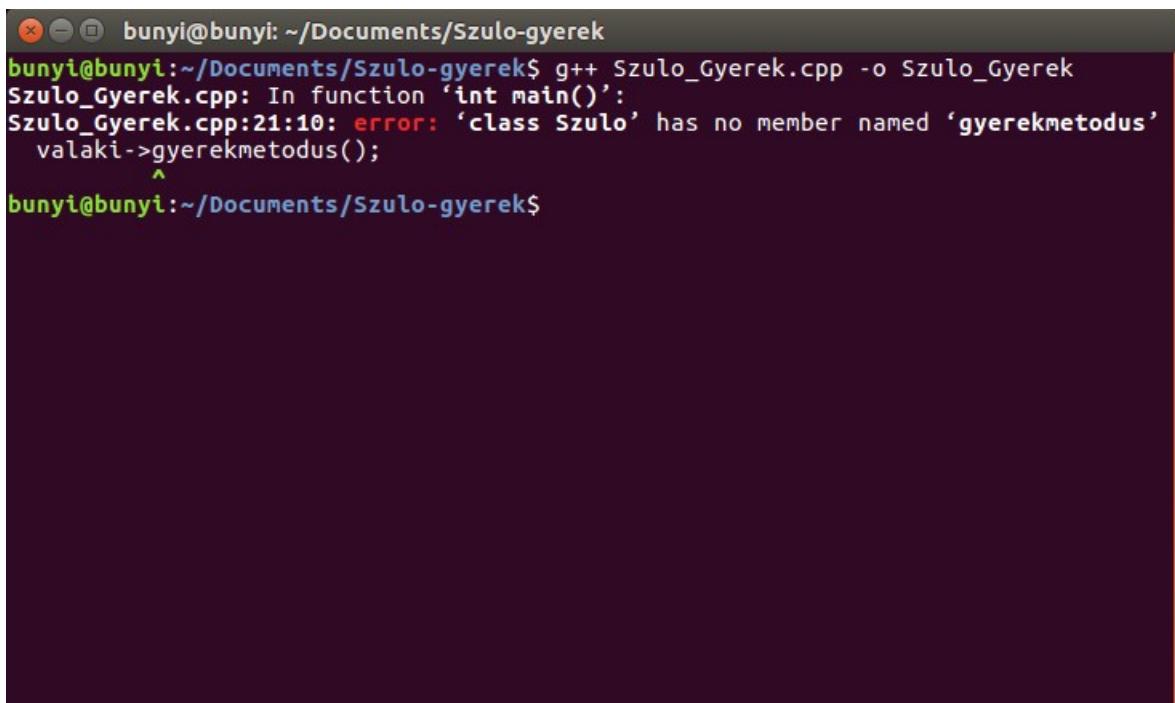
```
#include <iostream>

class Szulo {
public:
    void szulometodus() {
        std::cout << "Szulo vagyok!" << std::endl;
    }
};

class Gyerek : public Szulo {
public:
    void gyerekmetodus() {
        std::cout << "Gyerek vagyok!" << std::endl;
    }
};

int main() {
    Szulo* valaki = new Gyerek();

    valaki->szulometodus();
    valaki->gyerekmetodus(); //Nem látja a gyermekmetódusát!
}
```



The screenshot shows a terminal window with the following text:

```
bunyi@bunyi:~/Documents/Szulo-gyerek$ g++ Szulo_Gyerek.cpp -o Szulo_Gyerek
Szulo_Gyerek.cpp: In function 'int main()':
Szulo_Gyerek.cpp:21:10: error: 'class Szulo' has no member named 'gyerekmetodus'
    valaki->gyerekmetodus();
               ^
bunyi@bunyi:~/Documents/Szulo-gyerek$
```

Itt is ugyanúgy error kapunk, mivel hiába castoltuk a gyermeket szülővé az nem éri el a gyermek metódusát.

Anti OO

Össze kellett hasonlítani a BBP algoritmus kód futási idejét C, C++, Java és C# nyelven. Egy virtuális linux gépen futattam a kódokat. Ilyen eredményt kaptam:

	C	C++	Java	C#
10^6	3.051	2.823	2.575	2.617
10^7	34.460	33.332	28.978	30.589
10^8	385.376	386.714	338.139	353.068

4.1. táblázat. Összehasonlítás

Ahogy látszik a C nyelv volt a leglassabb. Ez várható volt, hisz ez a legöregebb nyelv a négy közül. Leggyorsabb volt a Java kód, amely a 10^8 pozícionál 10 másodperccel leelőzte a C#-t. Több oka is van, hogy a Java legyőzött mindenkit. Elsőnek lehet mondani, hogy a memória kiosztást sokkal jobban kezeli, mint a többi nyelv. Valamint a JVM jobban optimalizálja a metódus hívásokat. Futás időben dinamikus elemzést tud végezni, hogy mire van szükség és mire nem, így gyorsabb működést képes nyújtani, mint egy C++ fordítóprogram.

```
for (d = 1000000; d < 1000001; ++d)
{
    d16Pi = 0.0;

    d16S1t = d16Sj (d, 1);
    d16S4t = d16Sj (d, 4);
```

```
d16S5t = d16Sj (d, 5);
d16S6t = d16Sj (d, 6);

d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;

d16Pi = d16Pi - floor (d16Pi);

jegy = (int) floor (16.0 * d16Pi);

}
```

A d-t változtatva lehetett megadni a programnak, hogy 10^6 -on, 10^7 -en vagy 10^8 -on számjegyet szeretnénk megkapni. Java-ben ugyanígy csak ott paraméterként adtam át a számot.

Ciklomantikus komplexitás

Ebben a feladatban ki kellett számolnunk valamelyik programunk függvényeinek ciklomantikus komplexitását. Ezt én egy online program segítségével oldottam meg, a Lizard-dal. A BBP java kódját elemzte a program. Íme:

Try Lizard in Your Browser

.java

Analyse

```
public class BBP {
    String HexaJegyek;

    public BBP(int d) {

        double HexPi = 0.0;

        double S1 = Sj(d, 1);
        double S4 = Sj(d, 4);
        double S5 = Sj(d, 5);
        double S6 = Sj(d, 6);
```

Egyszerűen csak ki kell választanunk a forráskód nyelvét, majd beillesztenünk magát a kódot.

Code analyzed successfully.				
File Type	.java	Token Count	401	NLOC
Function Name	NLOC	Complexity	Token #	Parameter #
BBP::BBP	22	3	196	
BBP::toString	3	1	8	
BBP::Sj	6	2	70	
BBP::n16modk	17	5	87	
BBP::main	3	1	22	

A végeredményen a számok minél kisebbek annál jobb, hiszen ha túl bonyolultak a függvények nehezen olvasható a program.

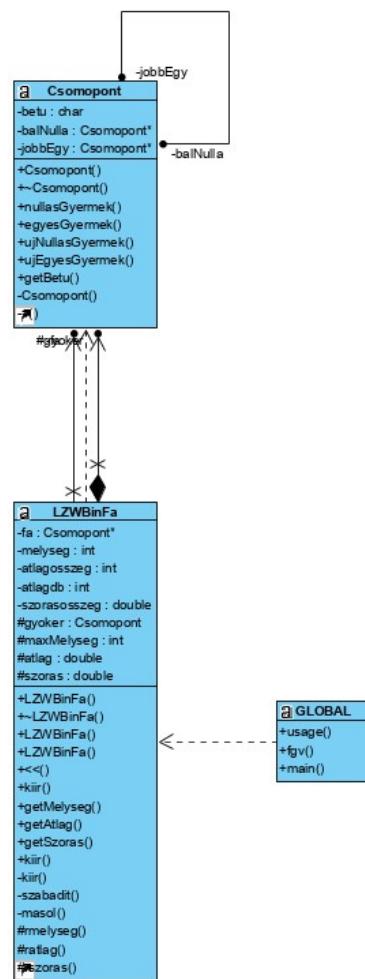
Számítása a gráfelméleten alapul. A forráskód alapján határozza meg az egyes függvények ciklomantikus komplexitását. Ez a független utak számát jelenti, hogy a program mennyire bonyolult vezérlési szempontból. Két út akkor számít függetlennek, ha minden kettőben van olyan pont, amely nem eleme a másiknak.

5. fejezet

Helló, Mandelbrot!

Reverse engineering UML osztálydiagram

A feladatban az első védési program C++ kódjából kellett UML osztálydiagramot generálnunk. Az UML egy grafikus modellező nyelv, amely diagramokat tartalmaz. A diagramok dobozokat, szövegeket, ikonokat és vonalakat foglalja magába. Az UML generálásához a Visual Paradigm alkalmazást használtam. Nekem ezt a diagramot alkotta:



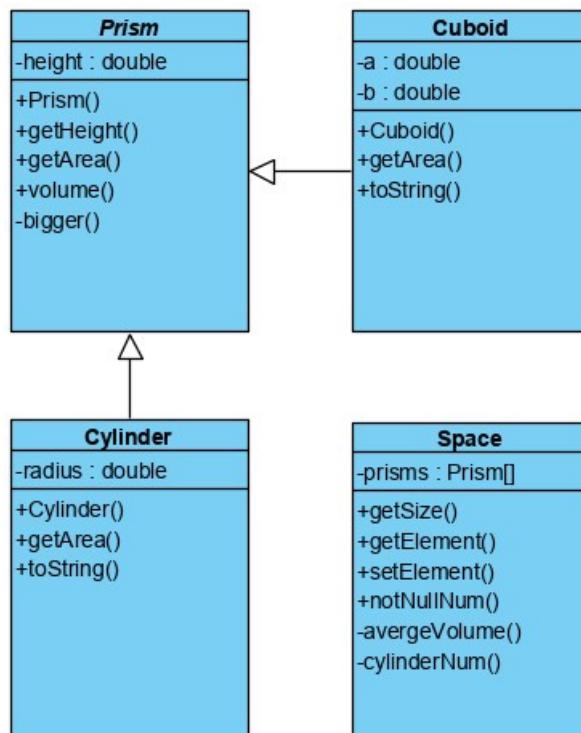
Az ábrán a kék dobozok az osztályok, tehát a Csomopont, az LZWBinFa és a GLOBAL. Közvetlenül az osztályok neve alatt találhatóak a változók. Az elérhetőségüket egy -, + vagy egy # jelöli. A - jelenti a private-ot, a + a publicot és a # a protectedet. A változók után egy vízszintes vonallal elválasztva következnek az osztály metódusai. Az elérhetőség itt is ugyanúgy van jelölve. A törzse mindegyiknek üresen van hagyva, hiszen ez csak egy modell, nem kell kódot írni bele, sőt nem is lehet.

Az asszociáció társítást jelent. A legegyszerűbb fajtája az osztályok között rajzolt egyszerű vonal. Azt jelenti valamilyen kapcsolat áll fent a két osztály között. A vonal végén ha nyíl van az az asszociáció irányát, navigálhatóságát jelzi. Ha nincs nyíl akkor a kapcsolat két irányú. Egy X-el jelöljük a nyílat, ha az asszociáció nem navigálható.

Az asszociációt belül megkülönböztetjük a tartalmazást, amelynek két fajtája van, gyenge és erős. A gyenge tartalmazást nevezzük aggregációt. Ezt akkor mondhatjuk amikor a rész hozzátarozik valamihez, de létezik önállóan is. Jele az UML-ben az üres rombusz. Az erős tartalmazást nevezzük kompozíciót. Ekkor a tartalmazott nem létezhet önmagában, csak részként valamiben. Ennek jele a telített rombusz. Ez látható is a generált diagramon. Itt azt fejezi ki, hogy a Csomopont osztály az LZWBinFa osztály része és nem létezhet nélküle.

Forward engineering UML osztálydiagram

Itt az előző feladat ellenkezőjét kellett végrehajtanunk, egy általunk létrehozott UML diagramból kellett forrást generálnunk. Visual Paradigm-ban dolgoztam. Létrehoztam 4 db osztályt: Prism, Cuboid, Cylinder és Space. A Prism osztályt absztraktnak jelöltetem ez látszik abból, hogy dőltbetűkkel van írva. A nyilak pedig azt jelentik, hogy a Cuboid és Cylinder osztály a Prism leszármazottja.



```

public abstract class Prism {

    private double height;
}

```

```
public Prism() {
    // TODO - implement Prism.Prism
    throw new UnsupportedOperationException();
}

public void getHeight() {
    // TODO - implement Prism.getHeight
    throw new UnsupportedOperationException();
}

public void getArea() {
    // TODO - implement Prism.getArea
    throw new UnsupportedOperationException();
}

public void volume() {
    // TODO - implement Prism.volume
    throw new UnsupportedOperationException();
}

private void bigger() {
    // TODO - implement Prism.bigger
    throw new UnsupportedOperationException();
}
}
```

```
public class Cuboid extends Prism {

    private double a;
    private double b;

    public Cuboid() {
        // TODO - implement Cuboid.Cuboid
        throw new UnsupportedOperationException();
    }

    public void getArea() {
        // TODO - implement Cuboid.getArea
        throw new UnsupportedOperationException();
    }

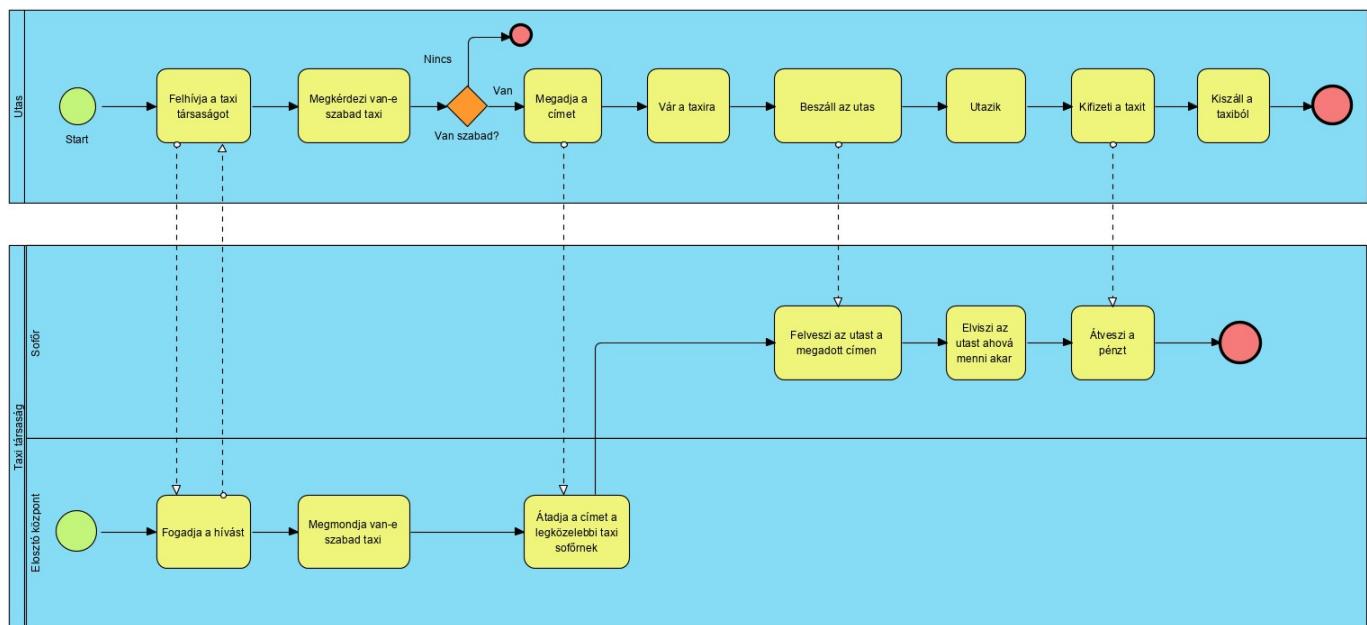
    public void toString() {
        // TODO - implement Cuboid.toString
        throw new UnsupportedOperationException();
    }
}
```

Ilyen kódokat generált a létrehozott diagramból. Ahogy látszik tényleg absztraktként hozta létre a Prism osztályt. A változókat is minden generálta. A metódus törzsekbe pedig írt, hogy implementálni kell a

metódusokat. Ez a fajta kódgenerálás inkább azoknak jó akik lusták megírni a kódot teljes egészében.

BPMN

A BPMN a Business Process Model and Notation rövidítése. Ez egy grafikus ábrázolás hasonló az UML-hez, az üzleti folyamatok modelljének meghatározására szolgál. A BPMN célja, hogy támogassa az üzleti folyamatok menedzselését, mind a műszaki felhasználók, mind az üzleti felhasználók számára. Egy szabványos jelölést biztosít, amely mindenkinél egyaránt könnyen érthető. Tehát egy közös nyelvként szolgál, áthaladva az üzleti folyamatok tervezése és megvalósítása között gyakran felmerülő kommunikációs rést.



Ez a példa egy hétköznapi folyamatot ír le, a taxi rendelés menetét. Az utas felhívja a taxi társaságot, ahol az elosztó központtal információt cserélnek. Az elosztó központ továbbítja a címet a legközelebbi sofőrnak. A sofőr felvesszi az utast elviszi a megadott helyre. Az utas kifizeti az útiköltséget és vége. Persze ez elég egyszerű példa, sokkal komolyabbakat is lehet modellezni.

BPEL Helló, Világ!

<https://youtu.be/eawsm8fb3iY>

Létre kellett hoznunk egy webszervert, amelynek egy stringet megadva ugyanazt a stringet dobja vissza. Egy 5 perces videóban meg is csináltam ezt. Azonban voltak előkészületek, mivel már egy depracated feladatról van szó. Először is hozzá kellett adnom az Eclipse-hez a BPEL pluginokat. Másodszor fel kellett telepíteni az Apache Tomcatet hozzáadva az ode kiegészítőt. Ezután a videó alapján megcsináltam a szervert, de szembesültem azzal, hogy nincs Web Services Explorer-em. Elég nehezen találtam meg, hogy az axis2 plugin tartalmazza ezt a funkciót.

A BPEL nyelvről még mondanék egy pár szót. A BPEL a Business Process Execution Language rövidítése. Ez az üzleti folyamatok leírására és webszerverrel kapcsolatos műveletekhez használják. A folyamatok a BPEL-ben az információt csak a webszolgáltatási felületek felhasználásával fogadják és továbbítják.

6. fejezet

Helló, Chomsky!

Encoding

A feladatban futtatni kellett a MandelbrotHalmazNagyító programot. Az volt benne a kihívás, hogy a forráskódban ékezes karakterek is szerepeltek és ha fordítottuk akkor ezt a hibát kaptuk:

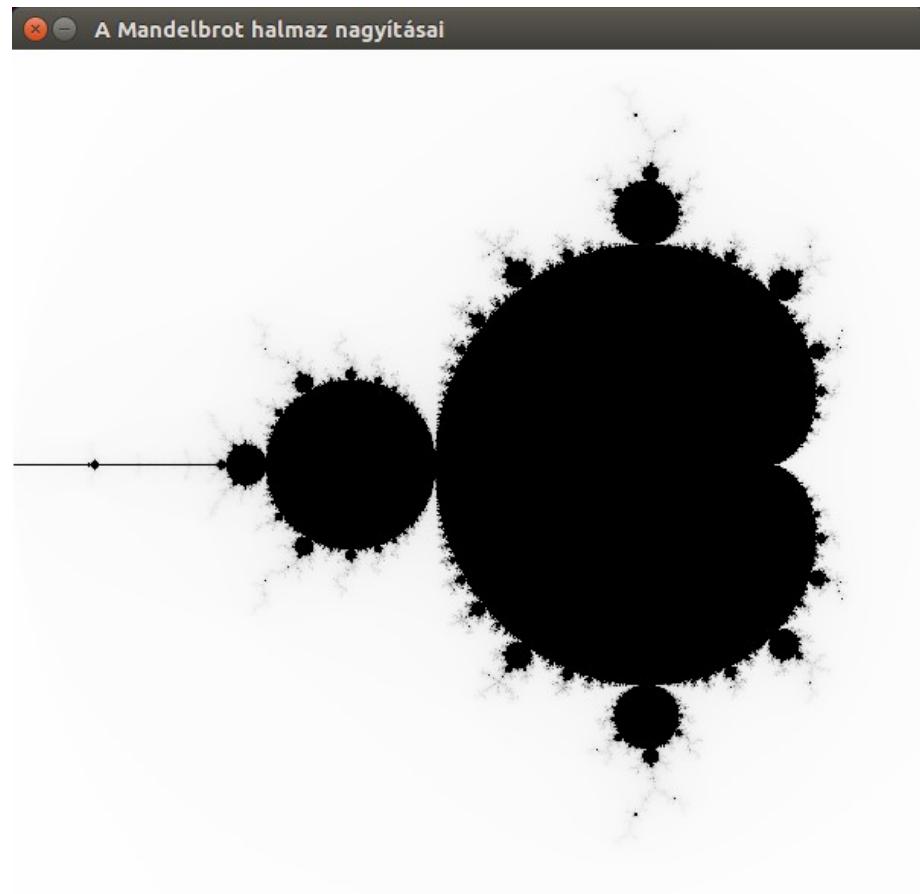
```
bunyi@bunyi: ~/Documents/Mandelbrot
bunyi@bunyi:~/Documents/Mandelbrot$ javac MandelbrotHalmazNagyító.java
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding
UTF-8
 * MandelbrotHalmazNagy t .java
          ^
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding
UTF-8
 * MandelbrotHalmazNagy t .java
          ^
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding
UTF-8
 * DIGIT 2005, Javat tan tok
          ^
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding
UTF-8
 * Batfai Norbert, nbatfai@inf.unideb.hu
          ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding
UTF-8
 * A Mandelbrot halmazt nagy t   s kirajzol  oszt ly.
          ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding
UTF-8
 * A Mandelbrot halmazt nagy t   s kirajzol  oszt ly.
```

Ahogy látszik a képen a kódolással lesz a hiba. Mivel az UTF-8-as kódolás számára nem találhatóak azok a karakterek amelyeket itt ? jelöl, ezek a kódban az ékezes betűk.

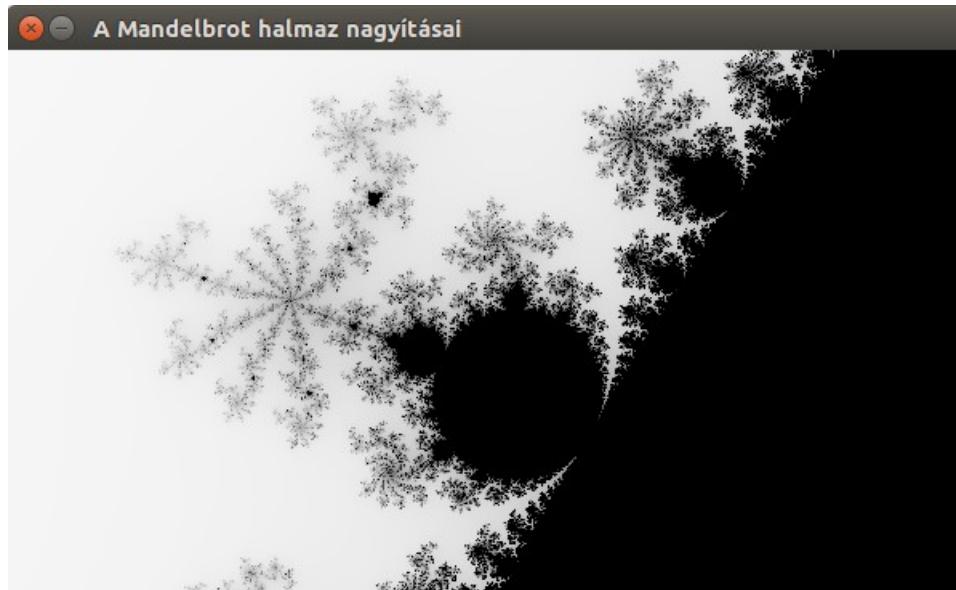
```
bunyi@bunyi: ~/Documents/Mandelbrot
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
  UTF-8
          // vizsgáljuk egy adott pont iterációt:
          ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
  UTF-8
          // vizsgáljuk egy adott pont iterációt:
          ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
  UTF-8
          // Az egérmutató pozíciója
          ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
  UTF-8
          // Az egérmutató pozíciója
          ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
  UTF-8
          // Az egérmutató pozíciója
          ^
100 errors
bunyi@bunyi:~/Documents/Mandelbrot$ javac -encoding "ISO-8859-2" MandelbrotHalma
zNagyító.java
bunyi@bunyi:~/Documents/Mandelbrot$
```

A megoldás az encoding kapcsoló volt rá a megfelelő kódolással. Kódolásnak pedig a Latin1 vagy Latin2 kellett megadni. Ezek a kódolások tartalmazzák az ékezetes betűket. Itt a Latin2 kellett használni, mert az tartalmazza az "ő" és "ű" is. Ennek a kódja pedig az ISO-8859-2 volt. Így már sikeresen fordult és futott is a program.

A program futás közben:



Ahogyan felnagyítja a képet:



|334d1c4^5

Olyan osztályt kellett írni, amely Leet chiperként működik. A Leet az egy másik angol ábécé, amit legfőképpen az interneten használnak. Az ASCII karakterek különféle kombinációját használja a latin betűk cseréjéhez. Legfőképpen az internetes fórumokon, chat-szobákban és online játékokban használják. Íme a kód:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Scanner;

public class LeetGood {

    private static char[] english_alphabet = {'a', 'b', 'c', 'd', 'e',
                                                'f', 'g', 'h', 'i', 'j',
                                                'k', 'l', 'm', 'n', 'o',
                                                'p', 'q', 'r', 's', 't',
                                                'u', 'v', 'w', 'x', 'y',
                                                'z', ' '};

    private static String[] leet_alphabet = {"4", "8", "<", "|)", "3",
                                             "|=", "9", "#", "1", "_|",
                                             "|<", "|_|", "|\\|/", "|\\|/",
                                             "0", "|2", "(,)", "|?", "$",
                                             "7", "|_|", "\\\/", "\\\|\\"|,
                                             "><", "|/", "2", " "};

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string to convert to leet speak:");
        String input = scanner.nextLine();
        String output = convertToLeet(input);
        System.out.println("Converted string: " + output);
    }

    private static String convertToLeet(String input) {
        StringBuilder output = new StringBuilder();
        for (char c : input.toCharArray()) {
            if (Character.isLetter(c)) {
                int index = english_alphabet.indexOf(c);
                if (index != -1) {
                    output.append(leet_alphabet[index]);
                } else {
                    output.append(c);
                }
            } else {
                output.append(c);
            }
        }
        return output.toString();
    }
}
```

```
static String readFile(String path, Charset encoding) throws ←
    IOException {
    byte[] encoded = Files.readAllBytes(Paths.get(path));
    return new String(encoded, encoding);
}

static String translator(String word) {
    char[] wordArray = word.toCharArray();
    StringBuilder res = new StringBuilder();

    for (int i = 0; i < wordArray.length; i++) {
        for (int j = 0; j < english_alphabet.length; j++) {
            if (wordArray[i] == english_alphabet[j]) {
                res.append(leet_alphabet[j]);
                break;
            }
        }
    }
    return res.toString();
}

public static void main(String[] args) throws IOException {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Kérem a fájl nevét: ");
    String fileName = scanner.nextLine();

    String fullFileContents = readFile(fileName, StandardCharsets.UTF_8) ←
    ;
    System.out.println("File beolvasva.");

    fullFileContents = fullFileContents.toLowerCase();
    fullFileContents = translator(fullFileContents);
    System.out.println("File lefordítva.");

    PrintWriter writer = new PrintWriter("leet.txt", "UTF-8");
    writer.print(fullFileContents);
    writer.close();
    System.out.println("A fordított szöveg az leet.txt-ben megtalálható ←
        .");

    scanner.close();
}
```

A feladat megoldásához tömböket alkalmaztam.

```

        'k', 'l', 'm', 'n', 'o',
        'p', 'q', 'r', 's', 't',
        'u', 'v', 'w', 'x', 'y',
        'z', ' '};

private static String[] leet_alphabet = {"4", "8", "<", "|)", "3",
                                         "|=", "9", "#", "1", "_|",
                                         "|<", "|_",
                                         "0", "2", "(,)", "|?", "$",
                                         "7", "|_|", "\\\/",
                                         "><", '/'', "2", " "};

```

Létrehoztam egy char és egy String típusú tömböt. A char típusú tömbbe kimentettem az angol abc betűit, a String típusúba pedig ugyanarra a tömb pozícióba a neki megfelelő leet abc betűt.

```

static String readFile(String path, Charset encoding) throws IOException {
    byte[] encoded = Files.readAllBytes(Paths.get(path));
    return new String(encoded, encoding);
}

```

A readFile() metódus egy szöveges fájl beolvasására szolgál.

```

static String translator(String word) {
    char[] wordArray = word.toCharArray();
    StringBuilder res = new StringBuilder();

    for (int i = 0; i < wordArray.length; i++) {
        for (int j = 0; j < english_alphabet.length; j++) {
            if (wordArray[i] == english_alphabet[j]) {
                res.append(leet_alphabet[j]);
                break;
            }
        }
    }
    return res.toString();
}

```

A translator() metódusban történik a lényeg, a szöveget amit beolvastam átadom neki, majd átalakítom és kimentem egy char típusú tömbbe. Példányosítok egy StringBuilder típusú változót, hogy egy szövegbe tudjam fűzni a leet abc betűit. Az első for ciklus végig lépked a tömb karakterein. A második for pedig az angol abc karakterein lépked. Ha a tömb karaktere és az angol abc karaktere megegyezik akkor egy szövegbe fűzöm az angol abc-nek megfelelő leet abc betűt. Ha beléptem az if metódusba egy break-el megtörök a belső for ciklust hiszen csak egyel lehet egyenlő így nem kell tovább néznem az abc betűit.

```

public static void main(String[] args) throws IOException {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Kérém a fájl nevét: ");
    String fileName = scanner.nextLine();

    String fullFileContents = readFile(fileName, StandardCharsets.UTF_8);
}

```

```
System.out.println("File beolvasva.");

fullFileContents = fullFileContents.toLowerCase();
fullFileContents = translator(fullFileContents);
System.out.println("File lefordítva.");

PrintWriter writer = new PrintWriter("leet.txt", "UTF-8");
writer.print(fullFileContents);
writer.close();
System.out.println("A fordított szöveg az leet.txt-ben megtalálható.");

scanner.close();
}
```

A main metóduson belül bekérem a beolvasandó fájl nevét, beolvasom a fájlt. Kisbetűssé alakítom, átadom a szöveget a translator() metódusnak. Majd kiírom egy külön fájlba a már átalakított szöveget.

Full screen

Egy Java-ban futó fullscreen-es programot kellett írnunk. Én egy bejelntkezési képernyőt írtam. Íme a kód:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;

public class Login {

    public static void main(String[] args) {

        //Create, set buttons

        //Login button
        JButton login = new JButton("Login");
        login.setBounds(50, 300, 275, 50);

        //Exit button
        JButton exit = new JButton("Exit");
        exit.setBounds(375, 300, 275, 50);
        exit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });

        //Create, set labels
```

```
//User name label
JLabel username = new JLabel("Username:");
username.setBounds(50, 50, 200, 75);
username.setFont(new Font("Arial", Font.PLAIN, 40));

//Password label
JLabel password = new JLabel("Password:");
password.setBounds(50, 150, 200, 75);
password.setFont(new Font("Arial", Font.PLAIN, 40));

//Create, set text fields

//User text field
JTextField userText = new JTextField();
userText.setBounds(275, 55, 375, 60);

//Pass text field
JTextField passText = new JTextField();
passText.setBounds(275, 155, 375, 60);

//Create, set panel
JPanel panel = new JPanel();

Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

//Set panel
int width = 700;
int height = 400;
panel.setBounds((dim.width-width)/2,
                (dim.height-height)/2, width, height);
panel.setLayout(null);

//Add buttons,labels,text fields to panel
panel.add(login);
panel.add(exit);
panel.add(username);
panel.add(password);
panel.add(userText);
panel.add(passText);

//Create, set frame
JFrame frame = new JFrame();
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
//set width, height full
frame.setUndecorated(true); // disable decorations on frame
frame.getContentPane().setBackground(new Color(87, 147, 77));
frame.setVisible(true); //show window
frame.setLayout(null);
```

```
    //Add panel to frame
    frame.add(panel);
}
}
```

Nézzük meg részenként a kódot.

```
//Create, set buttons

//Login button
JButton login = new JButton("Login");
login.setBounds(50, 300, 275, 50);

//Exit button
JButton exit = new JButton("Exit");
exit.setBounds(375, 300, 275, 50);
exit.addActionListener(new ActionListener() {
@Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
```

Ennél a résznél létrehoztam a Login és Exit gombokat és megadtam az Exit gombnak, hogy kilépj a programból, ha rákattintanak.

```
//Create, set labels

//User name label
JLabel username = new JLabel("Username:");
username.setBounds(50, 50, 200, 75);
username.setFont(new Font("Arial", Font.PLAIN, 40));

//Password label
JLabel password = new JLabel("Password:");
password.setBounds(50, 150, 200, 75);
password.setFont(new Font("Arial", Font.PLAIN, 40));
```

Címkek létrehozása. Ezek tartalmazzák a username és password szöveget.

```
//Create, set text fields

//User text field
JTextField userText = new JTextField();
userText.setBounds(275, 55, 375, 60);

//Pass text field
JTextField passText = new JTextField();
passText.setBounds(275, 155, 375, 60);
```

Létrehozom a mezőt ahol a user írhat.

```
//Create, set panel
JPanel panel = new JPanel();

Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

//Set panel
int width = 700;
int height = 400;
panel.setBounds((dim.width-width)/2, (dim.height-height)/2, width, height);
panel.setLayout(null);
```

Egy panel létrehozása, ami magába fogja foglalni a gombokat, címkkéket és text fieldeket.

```
//Add buttons,labels,text fields to panel
panel.add(login);
panel.add(exit);
panel.add(username);
panel.add(password);
panel.add(userText);
panel.add(passText);
```

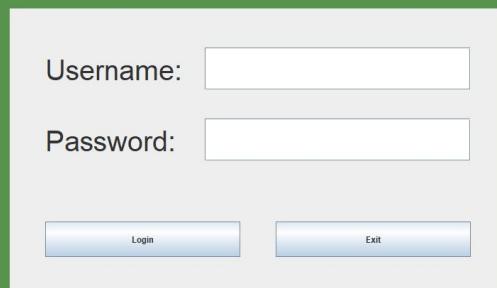
A panelhez hozzáadom a gombokat, címkkéket, text fieldeket.

```
//Create, set frame
JFrame frame = new JFrame();
frame.setExtendedState(Frame.MAXIMIZED_BOTH); //set width, height full
frame.setUndecorated(true); // disable decorations on frame
frame.getContentPane().setBackground(new Color(87, 147, 77));
frame.setVisible(true); //show window
frame.setLayout(null);

//Add panel to frame
frame.add(panel);
```

A lényegi rész pedig itt következik. Létrehozom a framet vagyis az ablakot. A setExtendedState-el maximálom a szélességét és hosszúságát az ablaknak. A setUndecorated(true)-val válik a program ténylegesen teljes képernyőssé. Ez "tünteti" el az ablak fejlécét, így fullscreen lesz a program.

Az utolsó sorokban már csak színezés, megjelenítés van és hozzáadjuk a panelt a framehez. Így néz ki a program:



Paszigráfia Rapszódia OpenGL full screen vizualizáció

A program első fordítása nem volt túl sikeres. Sok könyvtárat kellett leszednem ahhoz, hogy működjön: sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev. Ez a parancs telepítette nekem a glut könyvtárat ami hiányzott a fordításhoz. A fordításhoz a következő parancsot kellett használni: g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut -std=c++11. A futtatáshoz pedig ezt:

```
./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:1:  
0:2:2:0:1:1:1:3:2:1:0:2:0:2:1:1:1:2:3:0:1:1:1:1:0:3:3:0:1:0:2:1  
:0:1:0:2:2:0:0:0:1:3:1:0:1:3:2:1:0:2:0:3:3:0:1:0:2:1:0.
```

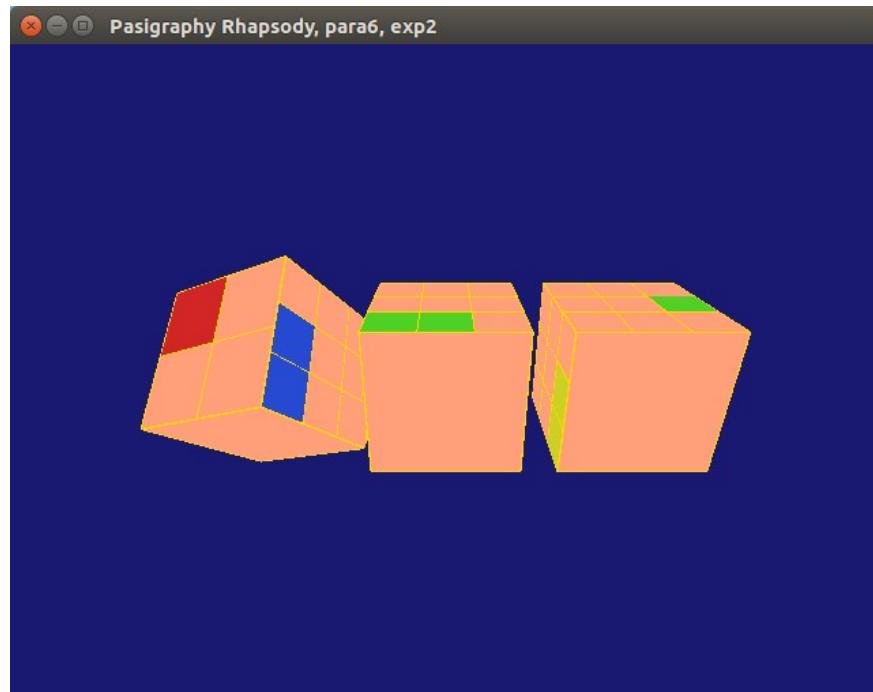
A program elindításakor rögtön feltűnt, hogy a kocka a nyilakkal ellentétes irányba mozdul. Egy kis keresgélés után rá lehetett jönni, hogy ebben a metódusban kell változtatni, hogy jól működjön. Csak annyit kellett, hogy ahol hozzáadok ott kivonok az értékből, ahol pedig kivonok ott hozzáadok az értékhez. Így az ellenkező irányba fog mozdulni minden.

```
void skeyboard ( int key, int x, int y )  
{  
    if ( key == GLUT_KEY_UP ) {  
        cubeLetters[index].rotx -= 5.0;  
    } else if ( key == GLUT_KEY_DOWN ) {  
        cubeLetters[index].rotx += 5.0;  
    } else if ( key == GLUT_KEY_RIGHT ) {  
        cubeLetters[index].roty += 5.0;  
    } else if ( key == GLUT_KEY_LEFT ) {  
        cubeLetters[index].roty -= 5.0;  
    } else if ( key == GLUT_KEY_PAGE_UP ) {
```

```
    cubeLetters[index].rotz -= 5.0;
} else if ( key == GLUT_KEY_PAGE_DOWN ) {
    cubeLetters[index].rotz += 5.0;
}

glutPostRedisplay();
}
```

A színezéssel játszadoztam még el. Ahhoz, hogy megváltoztassam a színét a glColor3f (R, G, B) metóduson kellett módosítanom a programban. Ez lett a végeredmény:



7. fejezet

Helló, Stroustrup!

JDK osztályok

Egy olyan programot kellett írni C++-ban boost használatával amely klistázza a JDK zip tartalmát. Íme a forráskód:

```
#include <iostream>
#include <vector>
#include <boost/filesystem.hpp>

using namespace std;
using namespace boost::filesystem;

int main(int argc, char *argv[])
{
    path p("SrcZip");
    if(!exists(p) || !is_directory(p)) {
        cout << p << " is not a path" << endl;
        return 1;
    }

    int i=0;
    recursive_directory_iterator begin(p), end;
    vector<directory_entry> v(begin, end);
    for(auto& f:v) {
        if(path(f).has_extension()) {
            cout << " -" << path(f) .filename() << endl;
            i++;
        } else {
            cout << f << endl;
        }
    }
    cout << "Összes fájl:" << i << endl;
}
```

Részenként elemezve a program:

```
#include <iostream>
#include <vector>
#include <boost/filesystem.hpp>

using namespace std;
using namespace boost::filesystem;
```

Először is includeolom a megfelelő osztályokat, közöttük a boost/filesystem.hpp, amely majd a path-al kapcsolatos műveletekben fog segíteni.

```
path p("SrcZip");
if(!exists(p) || !is_directory(p)) {
    cout << p << " is not a path" << endl;
    return 1;
}
```

Megadom pathnak a mappa nevét, amelyben a kicsomagolt src.zip fájl van. Valamint ellenőrzöm, hogy létezik-e megadott útvonal, ha nem kiíratom, hogy nincs ilyen útvonal.

```
int i=0;
recursive_directory_iterator begin(p), end;
vector<directory_entry> v(begin, end);
for(auto& f:v) {
    if(path(f).has_extension()) {
        cout << " -" << path(f).filename() << endl;
        i++;
    } else {
        cout << f << endl;
    }
}
cout << "Összes fájl:" << i << endl;
```

A recursive_directory_iterator egy directory_entry elemein az összes alkönyvtár bejegyzésén rekúrzióval iterál. minden könyvtárbejegyzést csak egyszer látogat meg. Egy vectorba elmentem az összes directory_entryt. Ezt átadom egy for-each ciklusnak. Egy if-el megvizsgálom, hogy az éppen vizsgált elérési útvonalnak van-e fájl a végén. Ha van beljebb húzom és kiíratom a fájl nevét, ha nincs akkor csak a mappa nevét íratom ki. Az i-vel pedig az összes fájlt számlálom meg. A program fordítása és futtatása:

The screenshot shows a terminal window with the following text:
bunyi@bunyi:~/Documents/JDK\$ g++ JdkClass.cpp -o JdkClass -std=c++11 -lboost_system -lboost_filesystem
bunyi@bunyi:~/Documents/JDK\$./JdkClass

A program végeredménye:

```
bunyi@bunyi:~/Documents/JDK
- "Sequencer.java"
- "MidiUnavailableException.java"
- "MidiChannel.java"
- "SoundbankResource.java"
"SrcZip/launcher"
- "jli_util.h"
- "java.h"
- "java_md.h"
- "splashscreen.h"
- "manifest_info.h"
- "java.c"
- "version_comp.c"
- "java_md.c"
- "wildcard.h"
- "parse_manifest.c"
- "wildcard.c"
- "defines.h"
- "splashscreen_stubs.c"
- "main.c"
- "version_comp.h"
- "emessages.h"
- "jli_util.c"
Összes fájl:7718
bunyi@bunyi:~/Documents/JDK$
```

Másoló-mozgató szemantika és Összefoglaló

Kódcsipeteken keresztül kellett összevetni a C++11 másoló és mozgató szemantikáját, valamint a mozgató konstruktort a mozgató értékkadásra kellett alapozni. Íme a teljes kód ami bemutatja mind a másoló konstruktort, másoló értékkadást, mozgató konstruktort és mozgató értékkadást:

```
#include <iostream>
#include <algorithm>

using namespace std;

class program{
private:
    int a;
    int* b;

public:
    program(int elem1, int& elem2) : a(elem1), b(new int){           // ←
        konstruktor 2 paraméterrel
        cout << "\tLefutott a default konstruktor!" << endl;
        *b = elem2;
    }

    program(const program &adott) {           //másoló konstruktor
        cout << "\tLefutott a másoló konstruktor!" << endl;
        a = adott.a;
        b = new int;
```

```
*b = *adott.b;
}

program& operator= (program &adott) {           //másoló értékkadás
    cout << "\tMásoló értékkadás történt!" << endl;
    a = adott.a;
    *b = *adott.b;
    return *this;
}

program(program && adott) {           //mozgató konstruktor
    cout << "\tLefutott a mozgató konstruktor!" << endl;
    a = 0;
    b = nullptr;
    *this = move(adott);

}

program& operator= (program && adott) {           //mozgató értékkadás
    cout << "\tMozgató értékkadás történt!" << endl;
    swap(b, adott.b);
    swap(a, adott.a);

    return *this;
}

void Print(){
    if(b!=NULL)
        cout << "Az a értéke: " << a << ", A b értéke " << *b << ", és ←
            b " << b << "-re mutat\n" << endl;
    else
        cout << "Az a értéke: " << a << ", A b értéke " << b << "\n" << ←
            endl;
}

~program() {           //destrukturátor
    cout << "\tLefutott a destrukturátor!" << endl;
    delete b;
}
};

int main(){

    int Nyolc = 8;
    int Kilenc = 9;

    cout << "Alap létrehozása 10, Nyolc értékkkel:" << endl;
    program Alap(10, Nyolc);
    cout << "Alap értékei:" << endl;
    Alap.Print();
}
```

```
cout << "\n\n-----Másoló konstruktor-----" << endl;
cout << "Alap_masolat létrehozása Alap alapjan:" << endl;
program Alap_masolat(Alap);
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Alap_masolat értékei" << endl;
Alap_masolat.Print();

cout << "\n\nUj1 létrehozása 20, Kilenc értékekkel:" << endl;
program Uj1(20, Kilenc);
cout << "Uj1 ertekei:" << endl;
Uj1.Print();

cout << "\n\n-----Másoló értékadás-----" << endl;
cout << "Alap_masolat = Uj1:" << endl;
Alap_masolat = Uj1;
cout << "Uj1 ertekei:" << endl;
Uj1.Print();
cout << "Alap_masolat ertekei:" << endl;
Alap_masolat.Print();

cout << "\n\n-----Mozgató konstruktor és értékadás-----" << endl;
cout << "Uj2 létrehozása mozgató konstruktorral:" << endl;
program Uj2(move(Alap));
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Uj2 ertekei:" << endl;
Uj2.Print();

cout << "\n\nProgram vége:" << endl;

return 0;
}
```

De nézzük elemezve a kódot.

```
private:
    int a;
    int* b;
```

A program class private részében először is van két változó. Egy alap int típusú ez lesz az "a". A másik egy pointer amit * jelölünk ez a "b".

```
program(int elem1, int& elem2) : a(elem1), b(new int){
    cout << "\tLefutott a default konstruktor!" << endl;
    *b = elem2;
}
```

A legelső egy default két paraméteres konstruktur. Itt csak annyi történik, hogy amikor ez meghívódik akkor az "a"-t egyenlővé teszi az első elemmel, a "b" viszont mivel pointer jelölni kell egy *-al, így mentjük el az értékébe az elem2-t. A kiíratás egy nyomkövető üzenet.

```
program(const program &adott) {
    cout << "\tLefutott a másoló konstruktor!" << endl;
    a = adott.a;
    b = new int;
    *b = *adott.b;
}
```

Ezután a sorban a másoló konstruktur következik. Ezt akkor használjuk amikor úgy inicializálunk egy osztályt, hogy egy már kész ugyanolyan típusú osztály példány értékeit másoljuk át. Ez a konstruktur annyira fontos a C++-ban, hogy ha nem hozunk létre ilyet, automatikusan létre fog jönni egy. Azonban ezek a fordító által létrehozott konstruktorkok csak sekély másolatot készítenek. Ez azt jelenti, hogy csak az eredeti objektumra való hivatkozást másolja és nem annak az értékét.

Ezért ha pointerekkel és referenciákkal dolgozunk csak maga a pointer másolódik, ekkor kell létrehoznunk egy saját mélymásoló konstruktort. A mélymásoló külön memóriát foglal le a másolt információk számára. Vagyis a forrás és a másolat különböznek. Ha a mélymásolásnál megváltoztatjuk a másolat értékét akkor a forrás nem változik, nem úgy mint a sekély másolásnál. Tehát az egyik memória helyen végrehajtott módosítások nem befolyásolják a másikat. Ha pedig dinamikus memóriát pointerek segítségével allokálunk, akkor kell a felhasználó által definiált másoló konstruktur, hogy minden objektum különböző memória-helyekre mutasson.

Egy ilyen konstruktur látható fent is. A paraméterében egy objektumra vonatkozó referenciát kell tartalmaznia másképpen végtelen ciklus jön létre. A törzsben jelenleg egy nyomkövető üzenet van, egy sima int típusú érték átmásolás, valamint a b pointerbe másoljuk a másolandó példány b értékét.

```
program& operator= (program &adott) {
    cout << "\tMásoló értékkopírozás történt!" << endl;
    a = adott.a;
    *b = *adott.b;
    return *this;
}
```

A másoló értékkopírozást akkor használjuk amikor egy osztály értékeit egy másik már létező osztály értékeibe szeretnénk másolni. Meghívása: osztaly1 =osztaly2 Ez is ugyanúgy mint a másoló konstruktornál automatikusan létrejön a fordításkor, de ugyanaz lesz a helyzet itt is csak sekély másolás történik, így ahhoz, hogy pointerekkel és referenciákkal tudjunk dolgozni itt is létre kell hozni egy saját másoló értékkopírozást.

Ebben az esetben mint a másoló konstruktornál van egy sima int érték átmásolás és a b pointerbe másoljuk a másolandó példány értékét.

```
program(program && adott) {
    cout << "\tLefutott a mozgató konstruktor!" << endl;
    a = 0;
    b = nullptr;
    *this = move(adott);
}
```

```
program& operator= (program && adott) {
    cout << "\tMozgató értékkadás történt!" << endl;
    swap(b, adott.b);
    swap(a, adott.a);

    return *this;
}
```

Míg a másoló konstruktor és másoló értékkadás célja, hogy másolatot készítsen egyik objektumról a másikra, addig a mozgató konstruktor és értékkadásé, hogy átruházza az értékeket egyikről a másik objektumra. Ez a művelet sokkal kevesebb erőforrást igényel, mint a másolás, mivel itt nem kell lemásolni az értékeket csak átadni neki. Eközben az eredeti elveszti a tartalmát és használhatatlanná válik.

A mozgató konstruktor és értékkadás definíálása analóg történik. A másoló szemantika esetén a paraméter egy konstans bal érték referencia, amíg a mozgató szemantikánál egy nem konstans jobb érték referenciáról beszélünk. Ezt láthatjuk is a programban mivel a referencia & jelrel van jelölve a paraméterben.

A mozgató értékkadás csak annyit tesz, hogy felcseréli a két példány tagjait a már beépített swap metódussal. A mozgató konstruktort pedig a mozgató értékkadásra kellett alapozni. Ez úgy történik, hogy amikor meghívódik a konstruktor akkor amelyikbe mozgatni fogjuk az értékeket annak először kinullázzuk az értékeit, majd a `*this =move(adott);` sorral meghívjuk a másoló értékkadás függvényt, ez felcseréli a példányok értékeit, így a forrás kapja a nullákat és a másik a forrás értékeit.

```
void Print() {
    if(b!=NULL)
        cout << "Az a értéke: " << a << ", A b értéke " << *b << ", és b " ←
            << b << "-re mutat\n" << endl;
    else
        cout << "Az a értéke: " << a << ", A b értéke " << b << "\n" << ←
            endl;
}

~program() {
    cout << "\tLefutott a destruktur!" << endl;
    delete b;
}
```

A program osztályban ezután csak egy print metódus van, hogy szebben láthassuk mi is történik. Az "a" értékét írjuk ki, majd a "b" értékét, majd a memóriacímét ahová a "b" mutat. Valamint kell még egy destruktur, ami törli a pointereket a program végén, ezzel memóriát szabadítva fel.

```
int Nyolc = 8;
int Kilenc = 9;

cout << "Alap létrehozása 10, Nyolc értékkkel:" << endl;
program Alap(10, Nyolc);
cout << "Alap ertekei:" << endl;
Alap.Print();
```

A main metódusban először létrehozzuk az Alap példányt amit másolni fogunk és kiíratjuk az értékeit.

```
cout << "\n\n-----Másoló konstruktor-----" << endl;
cout << "Alap_masolat létrehozása Alap alapjan:" << endl;
program Alap_masolat(Alap);
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Alap_masolat értékei" << endl;
Alap_masolat.Print();
```

Ezután lemásoljuk az Alap példányt az Alap_masolatba és kiíratjuk az értékeiket.

```
cout << "\n\nUj1 létrehozása 20, Kilenc értékekkel:" << endl;
program Uj1(20, Kilenc);
cout << "Uj1 ertekei:" << endl;
Uj1.Print();

cout << "\n\n-----Másoló értékadás-----" << endl;
cout << "Alap_masolat = Uj1:" << endl;
Alap_masolat = Uj1;
cout << "Uj1 ertekei:" << endl;
Uj1.Print();
cout << "Alap_masolat ertekei:" << endl;
Alap_masolat.Print();
```

Létrehozunk egy Uj1 nevű példányt, amibe másoló értékadással átmásoljuk az Alap_masolat értékeit.

```
cout << "\n\n-----Mozgató konstruktor és értékadás-----" << endl;
cout << "Uj2 létrehozása mozgató konstruktorral:" << endl;
program Uj2(move(Alap));
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Uj2 ertekei:" << endl;
Uj2.Print();
```

Már csak a mozgató konstruktor és értékadás maradt. Ezzel a sorral: `program Uj2(move(Alap));` híjuk meg a konstruktort. Mivel a beépített move függvény az Alapból jobb érték referenciát hoz létre, így tudjuk meghívni.

A program futás után:

```
bunyi@bunyi: ~/Documents/Copy_test
Alap létrehozása 10, Nyolc értékkal:
    Lefutott a default konstruktor!
Alap ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

-----
Másoló konstruktor-----
Alap_masolat létrehozása Alap alapjan:
    Lefutott a másoló konstruktor!
Alap ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

Alap_masolat értékei
Az a értéke: 10, A b értéke 8, és b 0x1cd5050-re mutat

Uj1 létrehozása 20, Kilenc értékkel:
    Lefutott a default konstruktor!
Uj1 ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5070-re mutat

-----
Másoló értékadás-----
Alap_masolat = Uj1:
    Másoló értékadás történt!
Uj1 ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5070-re mutat

Alap_masolat ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5050-re mutat

-----
Mozgató konstruktor és értékadás-----
Uj2 létrehozása mozgató konstruktornal:
    Lefutott a mozgató konstruktor!
    Mozgató értékadás történt!
Alap ertekei:
Az a értéke: 0, A b értéke0

Uj2 ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

Program vége:
    Lefutott a destruktur!
    Lefutott a destruktur!
    Lefutott a destruktur!
```

Ahogyan látszik a másoló konstruktornál az értékek ugyanazok a másolat példányánál, de a memória cím más. A másoló értékadásnál ugyanez látható. A mozgató konstruktornál és értékadásnál pedig az látszik, hogy a forrást nullára állítottuk és az Uj2 kapta meg az értékeket.

Ahogyan látszik, ha jól megtanuljuk az alapokat nem okozhat nehézséget ezeknek a konstruktornak és értékadásoknak a használata, sőt megkönnyíthetik az egyes programok működését ha megfelelően használjuk őket.

Változó argumentumszámú ctor

Egy olyan példát kellett készítenünk, amely egy képet tesz a Perceptron osztály bemenetére és nem egy értéket, hanem egy ugyanakkora méretű képet ad vissza. Először is nézzük meg a konstruktorunkat:

```
public:  
    Perceptron ( int nof, ... )  
    {  
        n_layers = nof;  
  
        units = new double*[n_layers];  
        n_units = new int[n_layers];  
  
        va_list vap;  
  
        va_start ( vap, nof );  
  
        for ( int i {0}; i < n_layers; ++i )  
        {  
            n_units[i] = va_arg ( vap, int );  
  
            if ( i )  
                units[i] = new double [n_units[i]];  
        }  
  
        va_end ( vap );  
  
        weights = new double**[n_layers-1];  
  
#ifndef RND_DEBUG  
    std::random_device init;  
    std::default_random_engine gen {init() };  
#else  
    std::default_random_engine gen;  
#endif  
  
        std::uniform_real_distribution<double> dist ( -1.0, 1.0 );  
  
        for ( int i {1}; i < n_layers; ++i )  
        {  
            weights[i-1] = new double *[n_units[i]];  
  
            for ( int j {0}; j < n_units[i]; ++j )  
            {  
                weights[i-1][j] = new double [n_units[i-1]];  
  
                for ( int k {0}; k < n_units[i-1]; ++k )  
                {  
                    weights[i-1][j][k] = dist ( gen );  
                }  
            }  
        }  
    }
```

```
        }
    }
}
```

Ahogy látható a konstruktor paraméter mezőjében egy nem szokványos megadás van. Perceptron (int nof, ...) Ezt nevezzük változó paraméter számú konstruktornak. Az első szám azt fogja megadni, hogy hány paraméter lesz még utána.

```
double* operator() ( double image [] )
{
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

#ifndef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

#else

        #pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
        {
            units[i][j] = 0.0;

            for ( int k = 0; k < n_units[i-1]; ++k )
            {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
            }

            units[i][j] = sigmoid ( units[i][j] );
        }
    }
#endif
}

for (int i = 0; i < n_units[n_layers - 1]; i++) {
    image[i] = units[n_layers - 1][i];
}

return image;
//return sigmoid ( units[n_layers - 1][0] );
}
```

Itt pedig az operator() függvény látható, amelyben csak annyit kellett megváltoztatni, hogy ne egy értékkel

térjen vissza, hanem egy tömbbel.

```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
#include <fstream>

int main (int argc, char **argv) {
    png::image<png::rgb_pixel> png_image (argv[1]);

    int size = png_image.get_width() * png_image.get_height();

    Perceptron* p = new Perceptron (3, size, 256, size);

    double* image = new double[size];

    for (int i {0}; i<png_image.get_width(); ++i)
        for (int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;

    double* newPNG = (*p) (image);

    for (int i = 0; i < png_image.get_width(); ++i)
        for (int j = 0; j < png_image.get_height(); ++j)
            png_image[i][j].green = newPNG[i*png_image.get_width() + j];

    png_image.write("output.png");

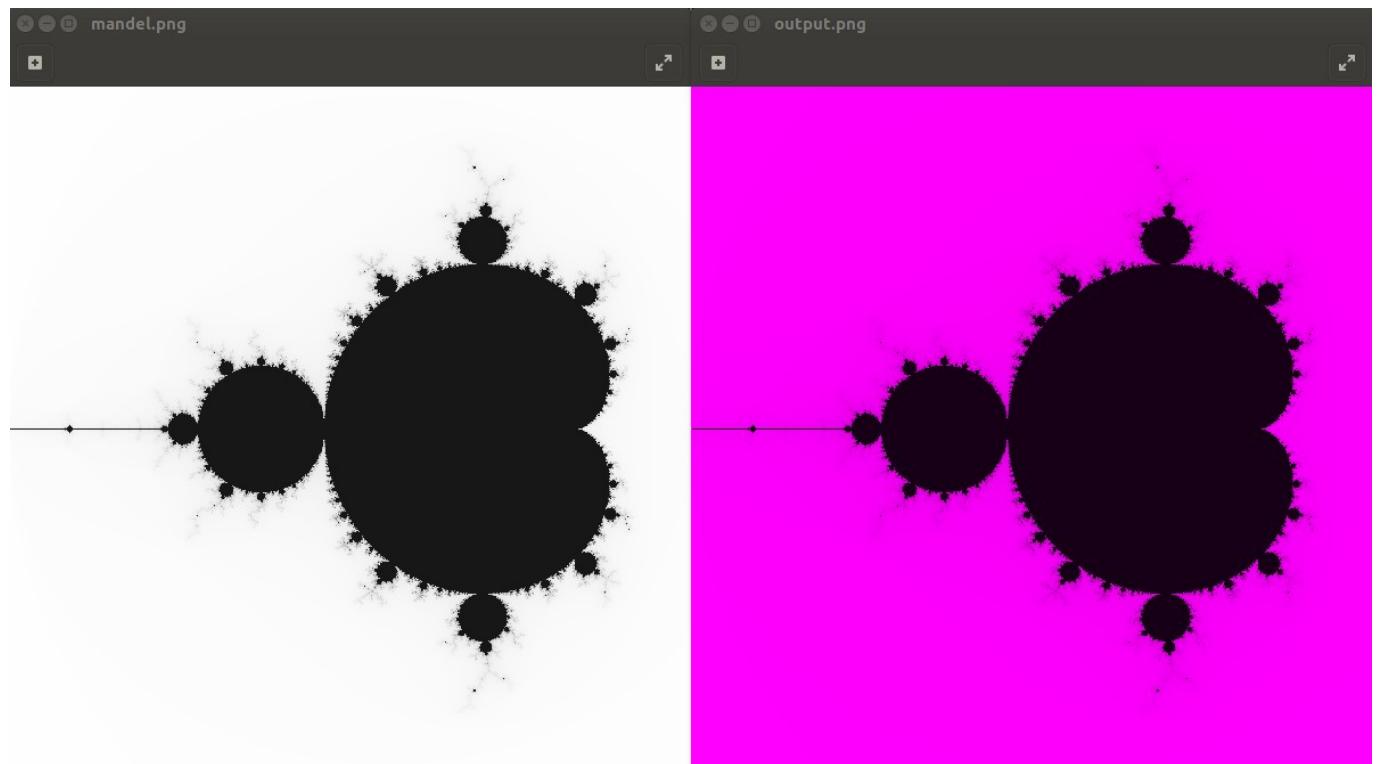
    delete p;
    delete [] image;
}
```

A main metódusban először beolvassuk a képet. Létrehozzuk a size-ot, amely a kép pixel méretét fogja tartalmazni. Majd példányosítjuk a Perceptron osztályt. Itt látszik a változó paraméterű konstruktor haszná. Mivel több szűrő szintet tudunk használni. Az első szám, jelen esetben ez 3 megadja, hogy hány szint lesz. A többi szám minden szinthez a unit-ok számát tartalmazza. Ezt mentjük, majd el az n_units tömmbe és létrehozunk még egy számára megfelelő méretű double tömböt is. Ezután feltöltjük a weights-et az értékeknek megfelelően. Ezután végig megyünk a kép szélességén és magasságán, a megfelelő piros értékeket kiírjuk az image tömbbe. A második for ciklussal felülírjuk a zöld értékeket. Végül kimentjük az új képet és felszabadítjuk a memóriát.

A program fordítása és futtatása:

```
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$ g++
mlp.hpp main.cpp -o perc -std=c++14 `libpng-config --ldflags`
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$ ./pe
rc mandel.png
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$ █
```

Az eredeti és az eredményül kapott kép:



8. fejezet

Helló, Gödel!

Gengszterek

A C++11 vezette be a lambda kifejezést, ez teszi lehetővé számunkra, hogy inline függvényeket írunk. Ezek rövid kód részletek, amelyeket nem használunk újra és nem érdemes megnevezni. Így kell definiálni:

```
[capture clause] (parameters) -> return-type
{
    definition of method
}
```

A lambda kifejezés visszatérítési típusát általában maga a fordító értékeli, így figyelmen kívül hagyhatjuk a `-> return-type` részt, kivéve néhány komplex esetben.

Az OOCWC-ben a 75. sorban lévő kód részletben van a lényeg:

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( Gangster x, ←
    Gangster y )
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
} );
```

Rendezésre az `std::sort` függvényt fogjuk használni, ami a benne szereplő lambda kifejezést fogja használni összehasonlításra. A sort metódus definíálása:

```
void sort (RandomAccessIterator first, RandomAccessIterator last, Compare ←
    comp);
```

Ezzel a metódussal a `gangsters` vector fogjuk rendezni. Paraméterként két `Gangster` objektumot vár. Ha az `x` gangster közelebb van az elkapott `cop` objektumhoz, mint az `y` gangster akkor igaz értéket térít vissza. Vagyis a rendezés után a vector elején a cophoz legközelebb álló gangsterek lesznek.

STL map érték szerinti rendezése

A feladat az volt, hogy egy mapet ne a kulcs szerint rendezzünk, hanem az érték szerint. Íme a kód:

```
#include <iostream>
#include <map>
#include <set>
#include <functional>

using namespace std;

int main() {

    map<string, int> Map;

    Map.insert(pair<string, int>("a", 55));
    Map.insert(pair<string, int>("b", 21));
    Map.insert(pair<string, int>("c", 98));
    Map.insert(pair<string, int>("d", 62));
    Map.insert(pair<string, int>("e", 120));
    Map.insert(pair<string, int>("f", 34));
    Map.insert(pair<string, int>("g", 42));
    Map.insert(pair<string, int>("h", 29));
    Map.insert(pair<string, int>("i", 27));
    Map.insert(pair<string, int>("j", 87));

    typedef function<bool(pair<string, int>, pair<string, int>)> Hasonlito;

    Hasonlito hasonlit =
        [](pair<string, int> else ,std::pair<string, int> masodik)
    {
        return else .second < masodik.second;
    };

    set<pair<string, int>, Hasonlito> sortedMap(Map.begin(), Map.end(), ←
        hasonlit);

    for (pair<string, int> elem : Map)
        cout << elem.first << " :: " << elem.second << endl;

    cout << "-----" << endl;

    for (pair<string, int> elem : sortedMap)
        cout << elem.first << " :: " << elem.second << endl;
}
```

Nézzük a kódot részekre bontva.

```
map<string, int> Map;

Map.insert(pair<string, int>("a", 55));
Map.insert(pair<string, int>("b", 21));
Map.insert(pair<string, int>("c", 98));
Map.insert(pair<string, int>("d", 62));
```

```
Map.insert(pair<string, int>("e", 120));
Map.insert(pair<string, int>("f", 34));
Map.insert(pair<string, int>("g", 42));
Map.insert(pair<string, int>("h", 29));
Map.insert(pair<string, int>("i", 27));
Map.insert(pair<string, int>("j", 87));
```

Először is létrehozunk egy mapet, ahol a kulcs string típusú, az érték pedig int típusú lesz. A mapet feltöljük értékekkel a map osztály beépített insert függvényével.

```
typedef function<bool(pair<string, int>, pair<string, int>)> Hasonlito;

Hasonlito hasonlit =
    [](pair<string, int> elseo, std::pair<string, int> masodik)
{
    return elseo.second < masodik.second;
};
```

Ezután létrehozok egy Hasonlito osztályt ami két párt kér el és egy bool típust ad vissza. Ennek az osztálynak létrehozok egy hasonlit metódust amely attól függően, hogy a két pár érték eleméből melyik a nagyobb igazat vagy hamisat ad vissza.

```
set<pair<string, int>, Hasonlito> sortedMap(Map.begin(), Map.end(), ←
    hasonlit);
```

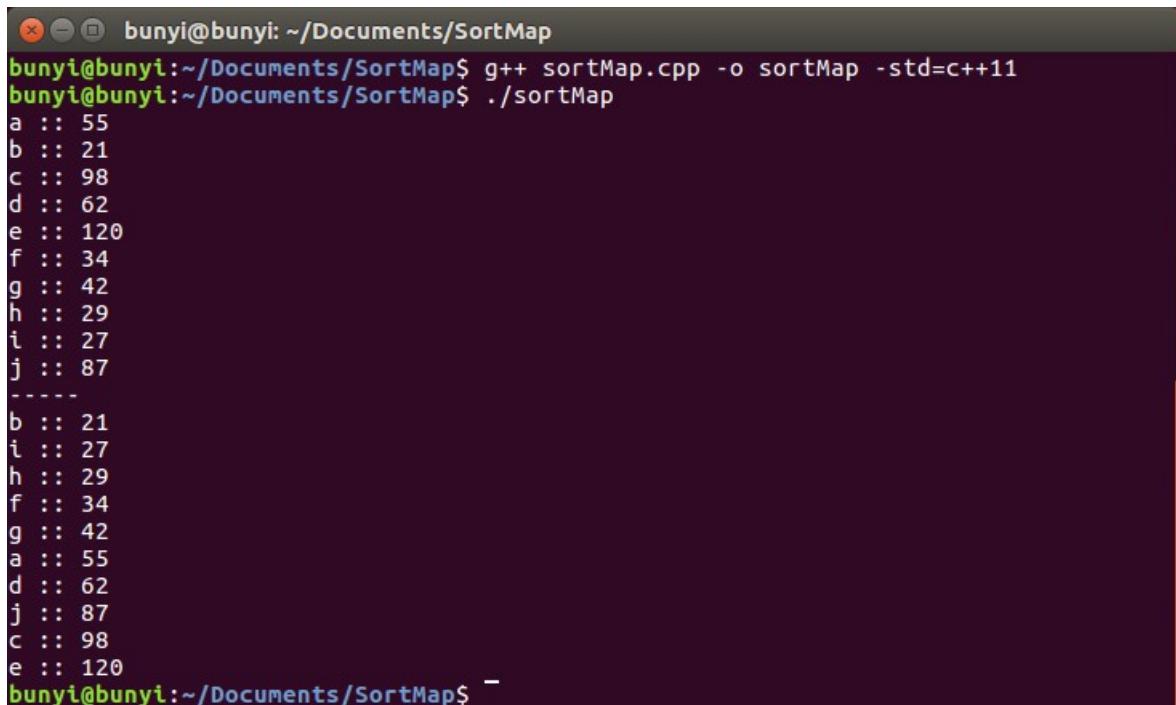
A seten belül történik a rendezés. A setnek meg kell adnom, hogy melyik osztályt használja a rendezéshez. Majd megadom a Map kezdő és végértékét és a metódust amelyet használnia kell a rendezéshez.

```
for (pair<string, int> elem : Map)
    cout << elem.first << " :: " << elem.second << endl;

cout << "-----" << endl;

for (pair<string, int> elem : sortedMap)
    cout << elem.first << " :: " << elem.second << endl;
```

Már csak a kiíratás marad amit egy for each ciklussal teszek meg. Elsőnek az eredeti Map-et íratom ki majd a már rendezettet. Ezt kaptam:



```
bunyi@bunyi:~/Documents/SortMap$ g++ sortMap.cpp -o sortMap -std=c++11
bunyi@bunyi:~/Documents/SortMap$ ./sortMap
a :: 55
b :: 21
c :: 98
d :: 62
e :: 120
f :: 34
g :: 42
h :: 29
i :: 27
j :: 87
-----
b :: 21
i :: 27
h :: 29
f :: 34
g :: 42
a :: 55
d :: 62
j :: 87
c :: 98
e :: 120
bunyi@bunyi:~/Documents/SortMap$ -
```

Alternatív Tabella rendezése

Ebben a feladatban a Comparable interface szerepére kellett rávilágítani. A Comparable interfacet használjuk a felhasználó által létrehozott osztályok rendezésére. Az interface a java.lang csomagban található és csak egyetlen metódust tartalmaz: `compareTo(Objektum)`. Így van a Csapat osztályban implementálva:

```
class Csapat implements Comparable<Csapat> {

    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Ezt az interfacet is mint ahogy a többet az implements kulcsszóval lehet implementálni. Valamint az interface kacsacsőrei között meg kell adni egy T generikus paramétert. Ennek az a célja, hogy így fordításkor egy erősebb fordításidéjű típusellenőrzést kapunk. Itt ez a típusparaméter a Csapat osztály lesz. A compareTo metódusnak egy Csapat objektumot kell megadni. Ha az objektum amihez hasonlítunk kisebb, mint az amit a zárójelen belül adunk meg akkor -1-el, ha nagyobb akkor 1-el, ha egyenlő akkor 0-val tér vissza.

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList(csapatok ↔
);
java.util.Collections.sort(rendezettCsapatok);
```

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the natural ordering of its elements. All elements in the list must implement the Comparable interface. Furthermore, all elements in the list must be *mutually comparable* (that is, e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the list).

A Java Dokumentációjában ahogy látszik, ahhoz, hogy a rendezés működjön a listákon minden elemnek implemetálni kell a Comparable interfacet. Ezt megkerestem a java.util.Collections forrásban is:

```
@SuppressWarnings("unchecked")
public static <T extends Comparable<? super T>> void sort(List<T> list) {
    list.sort(null);
}
```

Itt azt láthatjuk, hogy a T-nek vagy annak egy altípusának implementálni kell a Comparable interfacet. Az extends kulcsszó a T után itt általánosan használatos, azaz osztálynál extends lesz, interfacenél pedig implements.

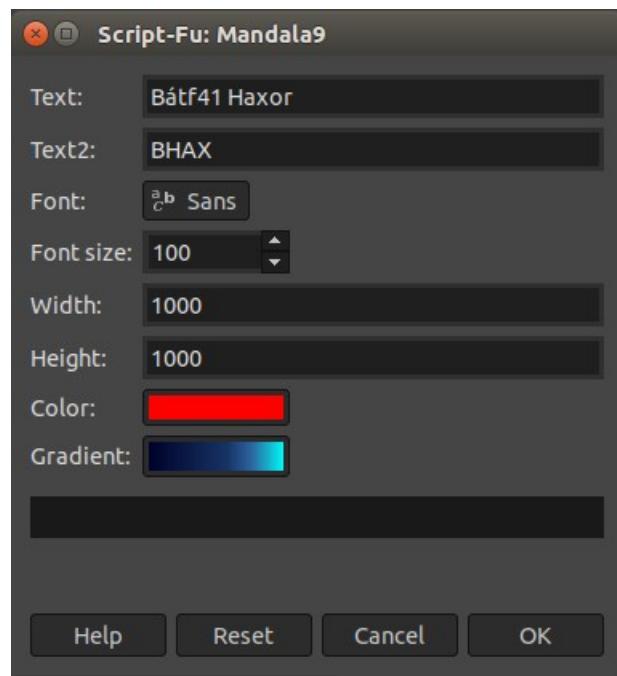
GIMP Scheme hack

Ebben a eladatban a Bátfai Norbert tanárúr által kiadott Gimp-es példát kellett feldolgozni.

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)
```

Amint belekükkantunk a forráskódba rögtön láthatjuk, hogy nem a megszokott programozással találkozunk. Nem infix alakban definiáljuk a műveleteket, hanem prefixben. Ennél a résznél például a define kulcsszóval létrehozunk egy metódust, amelynek 2 paramétere lesz.

Ahhoz, hogy a gimpben működjön ez a kód a gimp/227/.config/GIMP/2.10/scripts helyre kellet bemásolni. Ezután ha elindítjuk a Gimpet a létrehozásnál megjelenik a BHAX -> Mandala9. Ez az ablak ugrik fel ha rákattintunk:



Itt beállíthatjuk a szöveget amit elfog forgatni, a középső szöveget, a betűtípusot, a betűnagyságot, a mandala szélességét, hosszúságát, valamint a színezhetünk is.



9. fejezet

Helló, Valaki!

FUTURE tevékenység editor

A feladatban egy hibát kellett kijavítani a programban. Ha egy ideig tesztelgetjük a programot rögtön észre lehet venni, hogy csak egy Új altevékenységet lehet létrehozni.



```
bunyi@bunyi:~/media/sf_Feladatok/7.Het/1. FUTURE tevékenység editor/future-master/cs/F6$ javac ActivityEditor.java
bunyi@bunyi:~/media/sf_Feladatok/7.Het/1. FUTURE tevékenység editor/future-master/cs/F6$ java ActivityEditor
Cannot create City/Debrecen/Sport/Esport/DEAC-Hackers/Új altevénység
```

Tehát hiába próbálunk 2 vagy több altevékenységet létrehozni minden ezt a hibát kapjuk ami fent a konzolon látszik. Ezt a bugot viszonylag egyszerűen lehetett orvosolni.

```
public TextFieldTreeCell(javafx.scene.control.TextArea propsEdit) {
    this.propsEdit = propsEdit;
    javafx.scene.control.MenuItem subaMenuItem = new javafx.scene.control.MenuItem("Új altevékenység");//"New subactivity");
    addMenu.getItems().add(subaMenuItem);
    subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
        int i = 1;
        while(true) {
            java.io.File file = getTreeItem().getValue();

            java.io.File f = new java.io.File(file.getPath() + System.getProperty("file.separator") + "Új altevékenység" + i);

            if (f.mkdir()) {
                javafx.scene.control.TreeItem<java.io.File> newAct
                // = new javafx.scene.control.TreeItem<java.io.File>(f, new javafx.scene.image.ImageView(actIcon));
                = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
                getTreeItem().getChildren().add(newAct);
                break;
            } else {
                ++i;
                //System.err.println("Cannot create " + f.getPath());
            }
        }
    });
});
```

Egy végtelen ciklust kellett létrehozni a mappák létrehozásához egy számlálóval, így nem jöhet létre ugyanaz a mappa kétszer.



Így mostmár tetszőleges számú altevékenységet lehet létrehozni.

SamuCam

Azt a feladatot kaptuk, hogy a SamuCam projektben mutassunk rá a webkamera kezelésére. A program fordításához és futtatásához szükség van az opencv 2.4.9-es verziójára és a Qt valamilyen verziójára is. Valamint a program könyvtárába kell helyezni a lbpcascade_frontalface.xml-t.

```
SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = ↵
    144 )
: videoStream ( videoStream ), width ( width ), height ( height )
{
    openVideoStream();
}

SamuCam::~SamuCam ()
```

A program a parancssori futtatáskor megadott argumentumként kapja meg az IP címet ahonnan meg kell nyitnia a kamerát. Ha az eszközünkön lévő kamerát szeretnénk megnyitni akkor meg kell adni egy device ID-t, ez alapesetben 0 jelent.

```
void SamuCam::openVideoStream()
{
    videoCapture.open ( videostream );

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

Az openVideoStream()-ben a videoCapture.open() függvényel megnyitjuk a kamerát. A videoCapture.set()-tel pedig beállítjuk a szélességét, magasságát és az fps számot.

```
void SamuCam::run ()
{
    cv::CascadeClassifier faceClassifier;

    std::string faceXML = "lbpcascade_frontalface.xml"; // https://github.com ↵
        /Itseez/opencv/tree/master/data/lbpcascades

    if ( !faceClassifier.load ( faceXML ) )
    {
        qDebug() << "error: cannot found" << faceXML.c_str();
        return;
    }
```

A run() metóduson belül először példányosítunk egy CascadeClassifier-t ez alkalmas az Opencv-ben az objektumok felismerésére. Ezután megnyitjuk a már említett xml fájlt és ezt átadjuk a CascadeClassifier-nek, így válik lehetővé az arc felismerés. Majd lekezeljük a hibát ha nem sikerül betölteni az xmlt.

```
cv::Mat frame;
while ( videoCapture.isOpened() )
{
    QThread::msleep ( 50 );
    while ( videoCapture.read ( frame ) )
    {
        if ( !frame.empty() )
        {
            cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, cv::INTER_CUBIC );

            std::vector<cv::Rect> faces;
            cv::Mat grayFrame;

            cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
            cv::equalizeHist ( grayFrame, grayFrame );

            faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 4,
                cv::Size ( 60, 60 ) );

            if ( faces.size() > 0 )
            {
                cv::Mat onlyFace = frame ( faces[0] ).clone();

                QImage* face = new QImage ( onlyFace.data,
                    onlyFace.cols,
                    onlyFace.rows,
                    onlyFace.step,
                    QImage::Format_RGB888 );

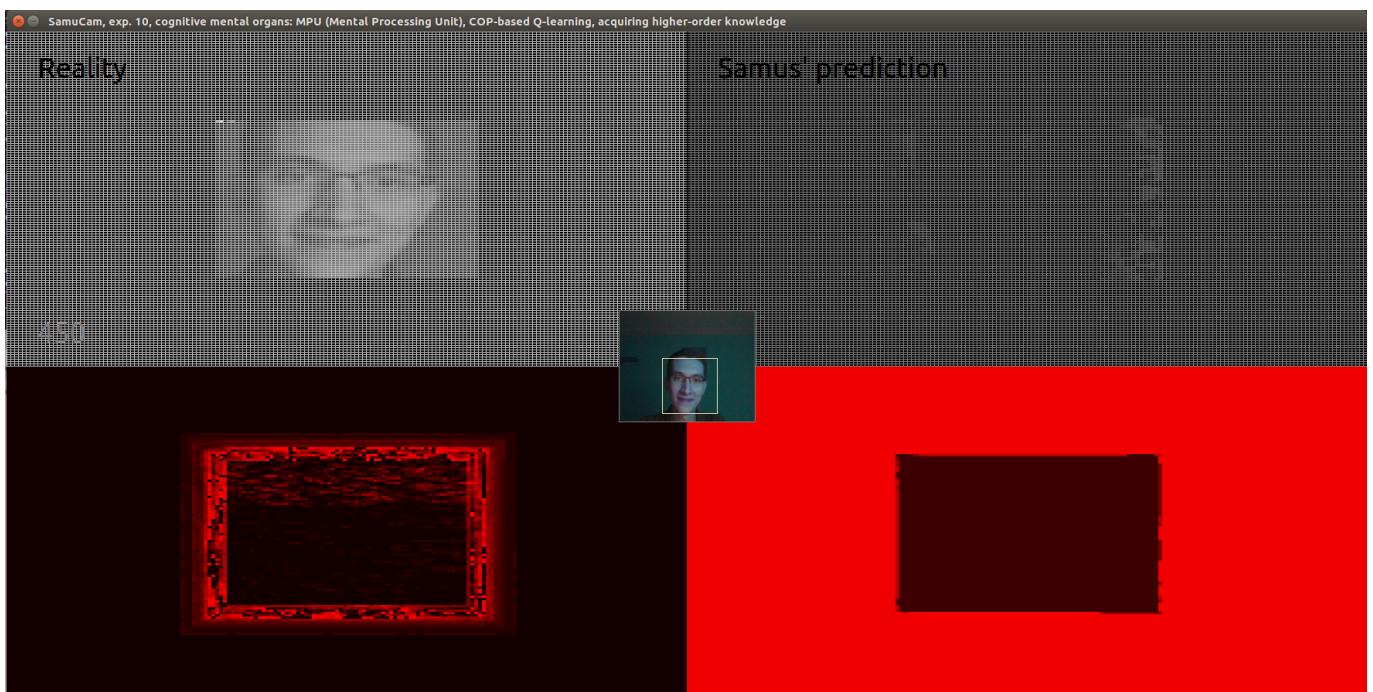
                cv::Point x ( faces[0].x-1, faces[0].y-1 );
                cv::Point y ( faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2 );
                cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230, 200 ) );
            }

            emit faceChanged ( face );
        }
    }
    QImage* webcam = new QImage ( frame.data,
        frame.cols,
```

```
frame.rows,  
frame.step,  
QImage::Format_RGB888 );  
  
emit webcamChanged ( webcam );  
  
}  
  
QThread::msleep ( 80 );  
  
}  
  
if ( ! videoCapture.isOpened() )  
{  
    openVideoStream();  
}  
  
}  
  
}  
}
```

Létrehozunk egy Mat típusú frame-et ebben tároljuk a kép adatokat. A run végén következik ez a két szép egymásba ágyazott while ciklus. Ez addig fog futni amíg a webkamera megvan nyitva ezt a videoCapture.isOpened() el ellenőrizzük, ami true-val tér vissza, ha megvan nyitva és false-al ha nincs. Ellenőrizzük, hogy a kép nem üres. A resize-al újra méretezzük a képet. A cvtColor-al pedig átalakítjuk szürkeskálásba a színes képet és az equalizeHist-el kiegyenlítjük a hisztogramot. A detectMultiScale-el keressük meg a fejet a képen ezt kimentjük a faces-be. Ezt adjuk át a SamuBrain-nek ami feldolgozza és frissíti a webcam képét és az ismétlődik addig, amíg be nem zárjuk a programot.

A program futás közben:



BrainB

Azt a feladatot kaptuk, hogy a BrainB-ben mutassuk be a Qt slot-signal mechanizmusát.

Ezt a mechanizmust objektumok közötti kommunikálásra lehet használni. Egy signal, vagyis egy jelet ad ki a program, ha egy bizonyos esemény bekövetkezik. A slot pedig egy olyan függvény amelyet a signal bekövetkezése esetén meg kell hívni. A connect függvénytelével lehet összekapcsolni egy signal és egy slotot. A két mechanizmus típus biztos ami annyit tesz, hogy a szignatúrájuknak egyezzenek kell. Valamint tetszőleges számú argumentumot felhevethetünk bármilyen típusból.

Így néz ki ez a BrainB-ben:

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ) ,
          this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ) ,
          this, SLOT ( endAndStats ( int ) ) );
```

Itt az történik, hogy ha a heroesChanged signal be fut akkor az az updateHeroes függvényt elindítja, ugyanez az endAndStat-nál.

```
void draw () {

    cv::Mat src ( h+3*heroRectSize, w+3*heroRectSize, CV_8UC3, cBg );

    for ( Hero & hero : heroes ) {

        cv::Point x ( hero.x-heroRectSize+dispShift, hero.y-heroRectSize+ ←
                      dispShift );
        cv::Point y ( hero.x+heroRectSize+dispShift, hero.y+heroRectSize+ ←
                      dispShift );

        cv::rectangle ( src, x, y, cBorderAndText );

        cv::putText ( src, hero.name, x, cv::FONT_HERSHEY_SIMPLEX, .35, ←
                      cBorderAndText, 1 );

        cv::Point xc ( hero.x+dispShift , hero.y+dispShift );

        cv::circle ( src, xc, 11, cCenter, CV_FILLED, 8, 0 );

        cv::Mat box = src ( cv::Rect ( x, y ) );

        cv::Mat cbox ( 2*heroRectSize, 2*heroRectSize, CV_8UC3, cBoxes );
        box = cbox*.3 + box*.7;

    }

    cv::Mat comp;

    cv::Point focusx ( heroes[0].x- ( 3*heroRectSize ) /2+dispShift, heroes ←
                      [0].y- ( 3*heroRectSize ) /2+dispShift );
```

```
cv::Point focusy ( heroes[0].x+ ( 3*heroRectSize ) /2+dispShift, heroes[0].y+ ( 3*heroRectSize ) /2+dispShift );
cv::Mat focus = src ( cv::Rect ( focusx, focusy ) );

cv::compare ( prev, focus, comp, cv::CMP_NE );

cv::Mat aRgb;
cv::extractChannel ( comp, aRgb, 0 );

bps = cv::countNonZero ( aRgb ) * 10;

//qDebug() << bps << " bits/sec";

prev = focus;

QImage dest ( src.data, src.cols, src.rows, src.step, QImage::Format_RGB888 );
dest=dest.rgbSwapped();
dest.bits();

emit heroesChanged ( dest, heroes[0].x, heroes[0].y );

}
```

Ebben a függvényben adja ki a signált. Ezzel: emit heroesChanged (dest, heroes[0].x, heroes[0].y);

```
void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &y )
{

    if ( start && !brainBThread->get_paused() ) {

        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + ( this->mouse_y - y ) * ( this->mouse_y - y );

        if ( dist > 121 ) {
            ++nofLost;
            noffound = 0;
            if ( nofLost > 12 ) {

                if ( state == found && firstLost ) {
                    found2lost.push_back ( brainBThread->get_bps() );
                }

                firstLost = true;

                state = lost;
                nofLost = 0;
                //qDebug() << "LOST";
            }
        }
    }
}
```

```
        //double mean = brainBThread->meanLost();
        //qDebug() << mean;

        brainBThread->decComp();
    }
} else {
    ++nofFound;
    nofLost = 0;
    if ( nofFound > 12 ) {

        if ( state == lost && firstLost ) {
            lost2found.push_back ( brainBThread ->get_bps() );
        }

        state = found;
        nofFound = 0;
        //qDebug() << "FOUND";
        //double mean = brainBThread->meanFound();
        //qDebug() << mean;

        brainBThread->incComp();
    }
}

pixmap = QPixmap::fromImage ( image );
update();
}
```

És ez a függvény fog lefutni mitán megkaphatja a signalt.

```
void BrainBThread::run()
{
    while ( time < endTime ) {

        QThread::msleep ( delay );

        if ( !paused ) {

            ++time;

            devel();

        }

        draw();
    }
}
```

```
    emit endAndStats ( endTime );  
}
```

Az endAndStats-nál ez a függvény fog signalozni. Ezzel:

```
void BrainBWin::endAndStats ( const int &t )  
{  
  
    qDebug() << "\n\n\n";  
    qDebug() << "Thank you for using " + appName;  
    qDebug() << "The result can be found in the directory " + statDir;  
    qDebug() << "\n\n\n";  
  
    save ( t );  
    close();  
}
```

És ezt a függvényt fogja elindítani.

Csak, hogy lássuk a programot futás közben:



10. fejezet

Helló, Lauda!

Port scan

A kivételkezelésre kellett rámutatni ebben a programkódban:

```
public class KapuSzkenner {

    public static void main(String[] args) {

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyeli");

                socket.close();

            } catch (Exception e) {

                System.out.println(i + " nem figyeli");

            }
    }
}
```

A kivételkezelés egy olyan programozási mechanizmus, amelynek során kezeljük a program futását megakadályozó eseményt. Ha egy kivétel megszakítja a normál végrehajtási folyamatot akkor általában végrehajt egy előre regisztrált kivételkezelőt. A végrehajtás részletei attól függnek, hogy hardveres vagy szoftveres kivételekről van-e szó. Néhány kivételt, általában szoftvereset annyira szépen lehet kezelní, hogy folytatni lehet a programot ott ahol megszakadt.

Ez a pár soros program arra jó, hogy megnézzük melyik portok vannak figyelve. A program úgy működik, hogy parancssori argumentumként megkapja a gépet és annak az 1024 alatti portjain megpróbál TCP

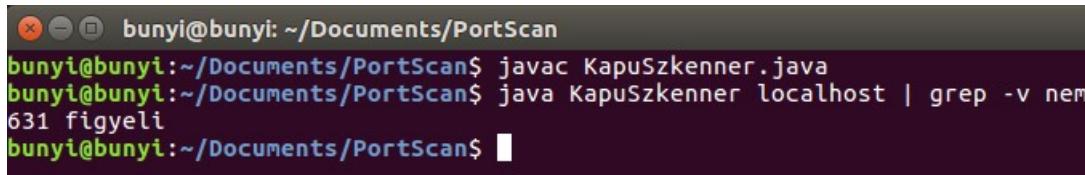
kapcsolatot kialakítani. Ha ez sikerül akkor a program kiírja a port nevét és azt, hogy figyeli, majd bezárja socketet. Azonban ha nem sikerül, akkor jön be a hibakeresés a képhez.

```
java.net.Socket socket = new java.net.Socket(args[0], i);
```

Ezzel a sorral próbál TCP kapcsolatot kialakítani, ha ez nem sikerül akkor IOException-t fog dobni. A catch ág ezt a kivételt elkapja és kiírja, hogy a port nincs figyelve. JDK forrásban így néz ki, látható, hogy ez a függvény két kivételt dobhat UnknownHostException-t és IOException-t.

```
public Socket(String host, int port)
    throws UnknownHostException, IOException
{
    this(host != null ? new InetSocketAddress(host, port) :
          new InetSocketAddress(InetAddress.getByName(null), port),
          (SocketAddress) null, true);
}
```

Futtatáskor csak egy portra figyelt:



```
bunyi@bunyi:~/Documents/PortScan
bunyi@bunyi:~/Documents/PortScan$ javac Kapuszkenner.java
bunyi@bunyi:~/Documents/PortScan$ java Kapuszkenner localhost | grep -v nem
631 figyeli
bunyi@bunyi:~/Documents/PortScan$
```

Android Játék

A feladatban egy Android játéket kellett írni. Én egy High-Low Game-t írtam. Íme a kód:

```
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
    private int theNumber;
    private int numberOfTries = 0;

    public void checkGuess() {
        String guessText = txtGuess.getText().toString();
        String message = "";
        try {
            int guess = Integer.parseInt(guessText);
            if (guess < theNumber) {
                message = guess + " is too low. Try again.";
                Toast.makeText(MainActivity.this,
                    6 - numberOfTries + " tries left", Toast. LENGTH_SHORT).show();
            } else if (guess > theNumber) {
                message = guess + " is too high. Try again.";
                Toast.makeText(MainActivity.this,
                    6 - numberOfTries + " tries left", Toast. LENGTH_SHORT).show();
            }
        }
```

```
        numberOfTries++;
        if (guess == theNumber) {
            message = guess + " is correct. You win after " + ←
                numberOfTries + " tries!";
            Toast.makeText(MainActivity.this, message, Toast. ←
                LENGTH_LONG).show();
            newGame();
        }
        if (numberOfTries == 7) {
            message = "You lose! The number was " + theNumber + ".";
            Toast.makeText(MainActivity.this, message, Toast. ←
                LENGTH_LONG).show();
            newGame();
        }
    } catch (Exception e) {
        message = "Enter a whole number between 1 and 100.";
    } finally {
        lblOutput.setText(message);
        txtGuess.requestFocus();
        txtGuess.selectAll();
    }
}

public void newGame() {
    theNumber = (int) (Math.random() * 100 + 1);
    numberOfTries = 0;
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtGuess = findViewById(R.id.txtGuess);
    btnGuess = findViewById(R.id.btnGuess);
    lblOutput = findViewById(R.id.lblOutput);

    newGame();
    btnGuess.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            checkGuess();
        }
    });
    txtGuess.setOnEditorActionListener(new TextView. ←
        OnEditorActionListener() {
        @Override
        public boolean onEditorAction(TextView v, int actionId, ←
            KeyEvent event) {
            checkGuess();
        }
    });
}
```

```
        return true;
    }
});
```

De nézzük részekre bontva a kódot.

```
private EditText txtGuess;
private Button btnGuess;
private TextView lblOutput;
private int theNumber;
private int numberOfTries = 0;
```

Először is létrehozom az osztály private változóit. A txtGuess-ben fogom megkapni a felhasználó által tippelt számot. A btnGuess-el fogom gombnyomásra ellenőrizni a megadott számot. Az lblOutput-tal pedig tudtára adom a felhasználónak, hogy alá vagy fölé lőtt vagy pontosan eltalálta a számot. A theNumberben fogom tárolni a helyes számot és a numberOfTries-al fogom számolni, hány lehetősége van még a felhasználónak kitalálni a számot.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtGuess = findViewById(R.id.txtGuess);
    btnGuess = findViewById(R.id.btnGuess);
    lblOutput = findViewById(R.id.lblOutput);

    newGame();
    btnGuess.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            checkGuess();
        }
    });
    txtGuess.setOnEditorActionListener(new TextView.OnEditorActionListener() {
        @Override
        public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
            checkGuess();
            return true;
        }
    });
}
```

A program futtásakor az onCreate metódus fog legelőször futni. Itt a txtGuess, btnGuess, lblOutput változókhöz hozzákapcsolom a GUI-ban a részüket. Majd hozzáadok két Listener-t. Az egyik a Guess! gombra fog figyelni, hogy ha megnyomják akkor ellenőrizze le a tippelt szám helyes-e. A setOnEditorActionListener azt figyeli, ha a szám begépelése után a pipára kattintunk akkor is ellenőrizze a helyességet. A newGame() metódussal pedig elindítjuk a játékot.

```
public void newGame() {  
    theNumber = (int) (Math.random() * 100 + 1);  
    numberOfTries = 0;  
}
```

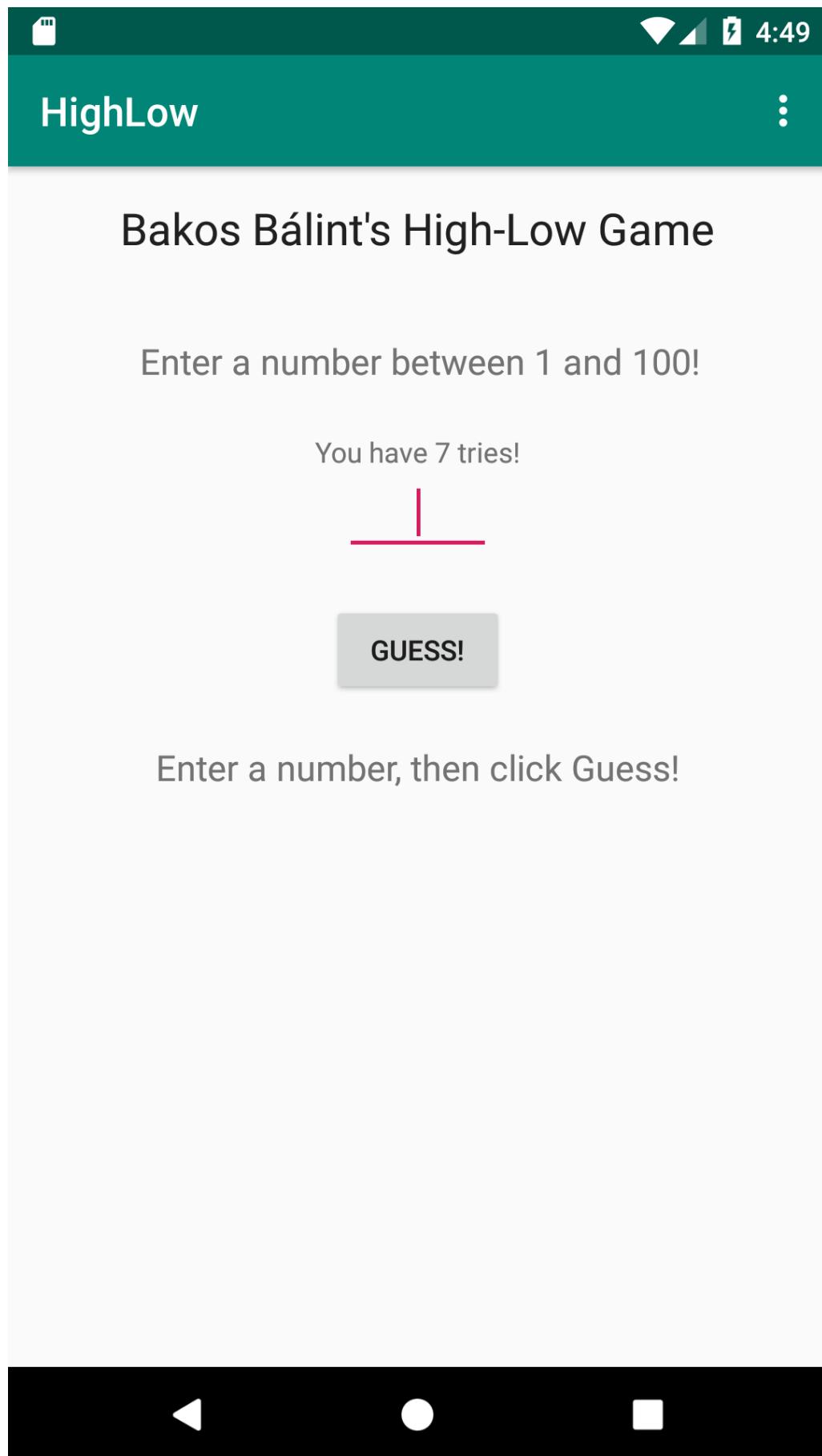
A newGame() a TheNumber-be kimenti a kitalálandó számot és lenullázza a probálkozások számát.

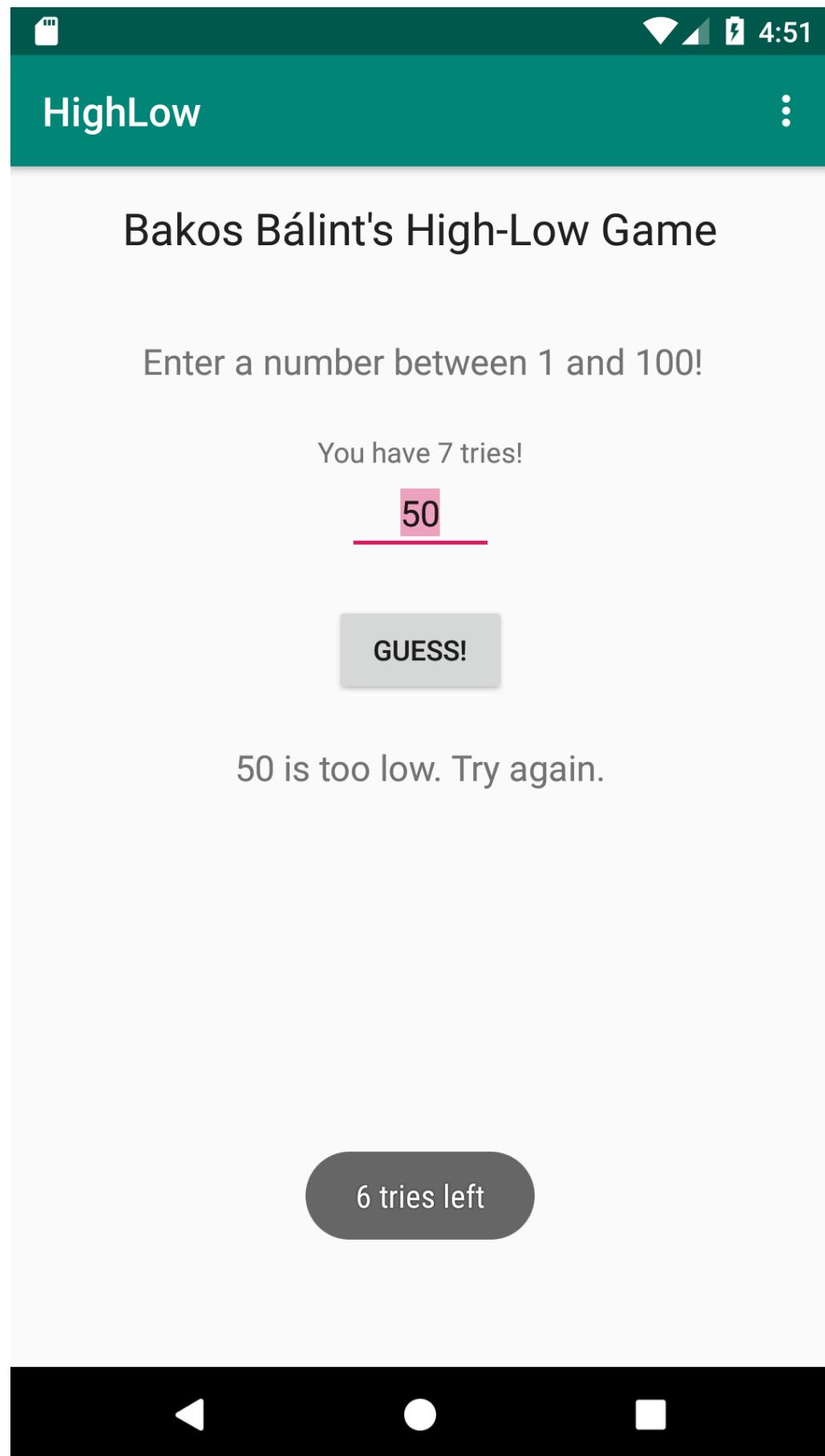
```
public void checkGuess() {  
    String guessText = txtGuess.getText().toString();  
    String message = "";  
    try {  
        int guess = Integer.parseInt(guessText);  
        if (guess < theNumber) {  
            message = guess + " is too low. Try again.";  
            Toast.makeText(MainActivity.this,  
                6 - numberOfTries + " tries left", Toast.LENGTH_SHORT). ←  
                show();  
        } else if (guess > theNumber) {  
            message = guess + " is too high. Try again.";  
            Toast.makeText(MainActivity.this,  
                6 - numberOfTries + " tries left", Toast.LENGTH_SHORT). ←  
                show();  
        }  
        numberOfTries++;  
        if (guess == theNumber) {  
            message = guess + " is correct. You win after " + numberOfTries ←  
                + " tries!";  
            Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG). ←  
                show();  
            newGame();  
        }  
        if (numberOfTries == 7) {  
            message = "You lose! The number was " + theNumber + ".";  
            Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG). ←  
                show();  
            newGame();  
        }  
    } catch (Exception e) {  
        message = "Enter a whole number between 1 and 100.";  
    } finally {  
        lblOutput.setText(message);  
        txtGuess.requestFocus();  
        txtGuess.selectAll();  
    }  
}
```

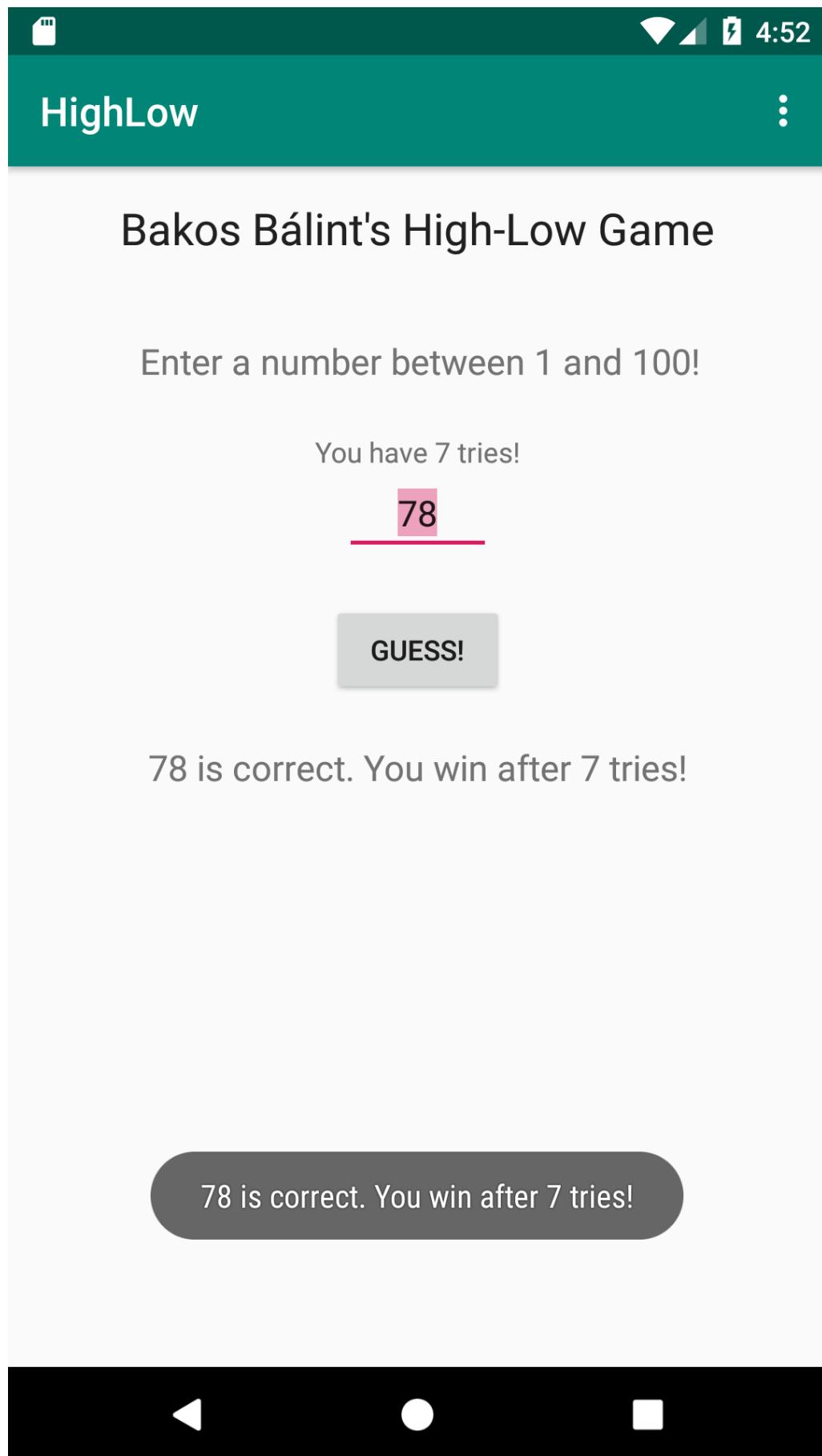
A checkGuess() metódusban a guessText String típusú változóba kimentem a felhasználó által megadott számot és egy message változót létrehozok, ebben fogok visszaadni valamilyen üzenetet a felhasználónak. A try-on belül nézem meg, hogy a szám nagyobb, kisebb vagy éppen egyenlő és ehhez képest adok vissza üzenetet. Ha a felhasználó nem számot vagy semmit sem adott meg az kivételt eredményez és a catch blokk

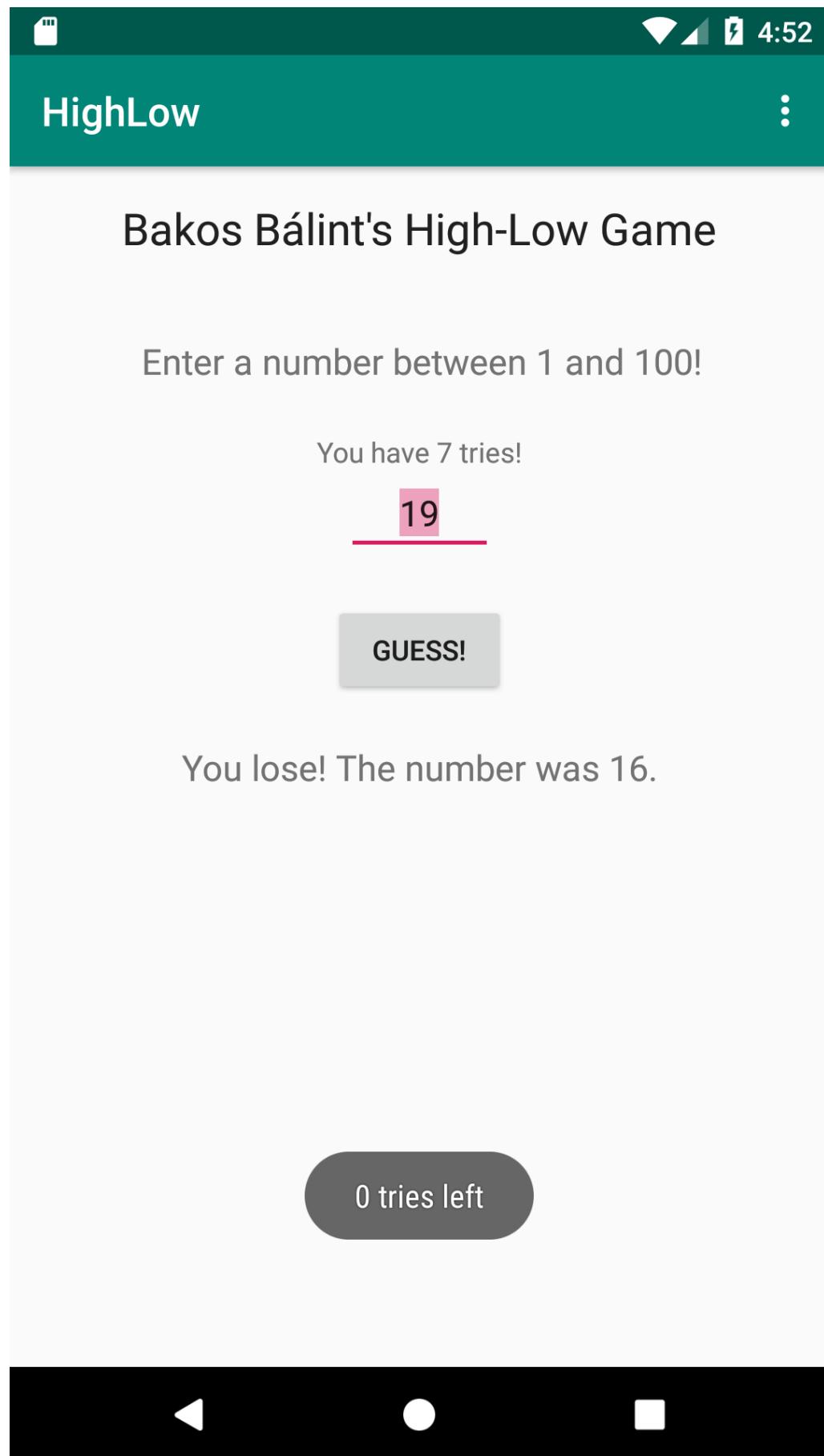
ezt elkapja. minden próbálkozás után kiírom egy popup üzenetben mennyi próbálkozás maradt még, ha elfogy a 7 próba a felhasználó vesztett és új játék kezdődik.

Képek a játékról:









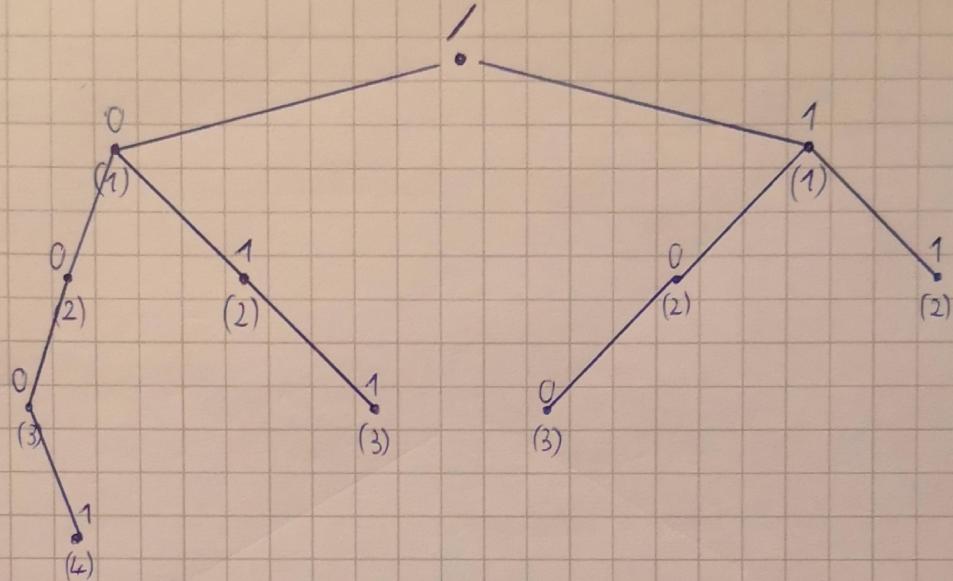
Junit teszt

Egy binfa példát kellett junit tesztbe beledolgozni. Először kiszámoltam az "abc" karakterek bináris kódját: 011000010110001001100011. Ezután megrajzoltam a fát, kiszámoltam az átlagot, a mélységet és a szórást.

Winform Informatikai Kft
 1145 Budapest, Uzsoki u. 36/A
 (+36 1) 273-0632, (+36 20) 999-5556
 info@winner.hu

WINNER
 az ügyvitel megoldása

0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1



Átlag:

$$\frac{4 + 3 + 3 + 2}{4} = 3$$

Szöns:

$$\begin{aligned}
 & \sqrt{\frac{1}{4-1} \cdot \left((4-3) \cdot (4-3) + \right.} \\
 & \quad \left. (3-3) \cdot (3-3) + \right. \\
 & \quad \left. (3-3) \cdot (3-3) + \right. \\
 & \quad \left. (2-3) \cdot (2-3) \right)} \\
 & = \sqrt{\frac{1}{3} \cdot (1+0+0+1)} = \\
 & = \sqrt{\frac{1}{3} \cdot 2} = \sqrt{\frac{2}{3}} = 0,81649
 \end{aligned}$$

Miután ezt megcsináltam JUnit teszt segítségével ellenőrizni kellett Junit teszttel, hogy helyesek-e az értékek amiket kézzel számoltam.

Mi is az a JUnit? A JUnit egy keretrendszer a Java programozási nyelvhez. A JUnit-ot egy osztály tesztelésére használják. minden tesztelés egy @Test annotációval kezdődik. Egy Assert típusú metódust kell használni, hogy összevessük a várható eredményt a valós eredménnyel.

Így teszteltem a Binfában a mélységet, átlagot és szórást:

```
import org.junit.jupiter.api.Test;

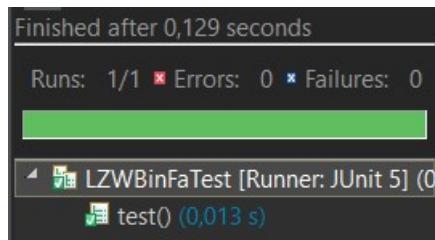
class LZWBinFaTest {
    LZWBinFa binfa = new LZWBinFa();

    @Test
    void test() {
        for (char c : "011000010110001001100011".toCharArray())
            binfa.egyBitFeldolg(c);

        org.junit.Assert.assertEquals(4, binfa.getMelyseg(), 0.0);
        org.junit.Assert.assertEquals(3, binfa.getAtlag(), 0.001);
        org.junit.Assert.assertEquals(0.81649, binfa.getSzoras(), 0.0001);
    }

}
```

Ahogy a képen látszik sikeresen tesztelte az értékeket és nem dobott hibát.



11. fejezet

Helló, Calvin!

MNIST

Deep MNIST

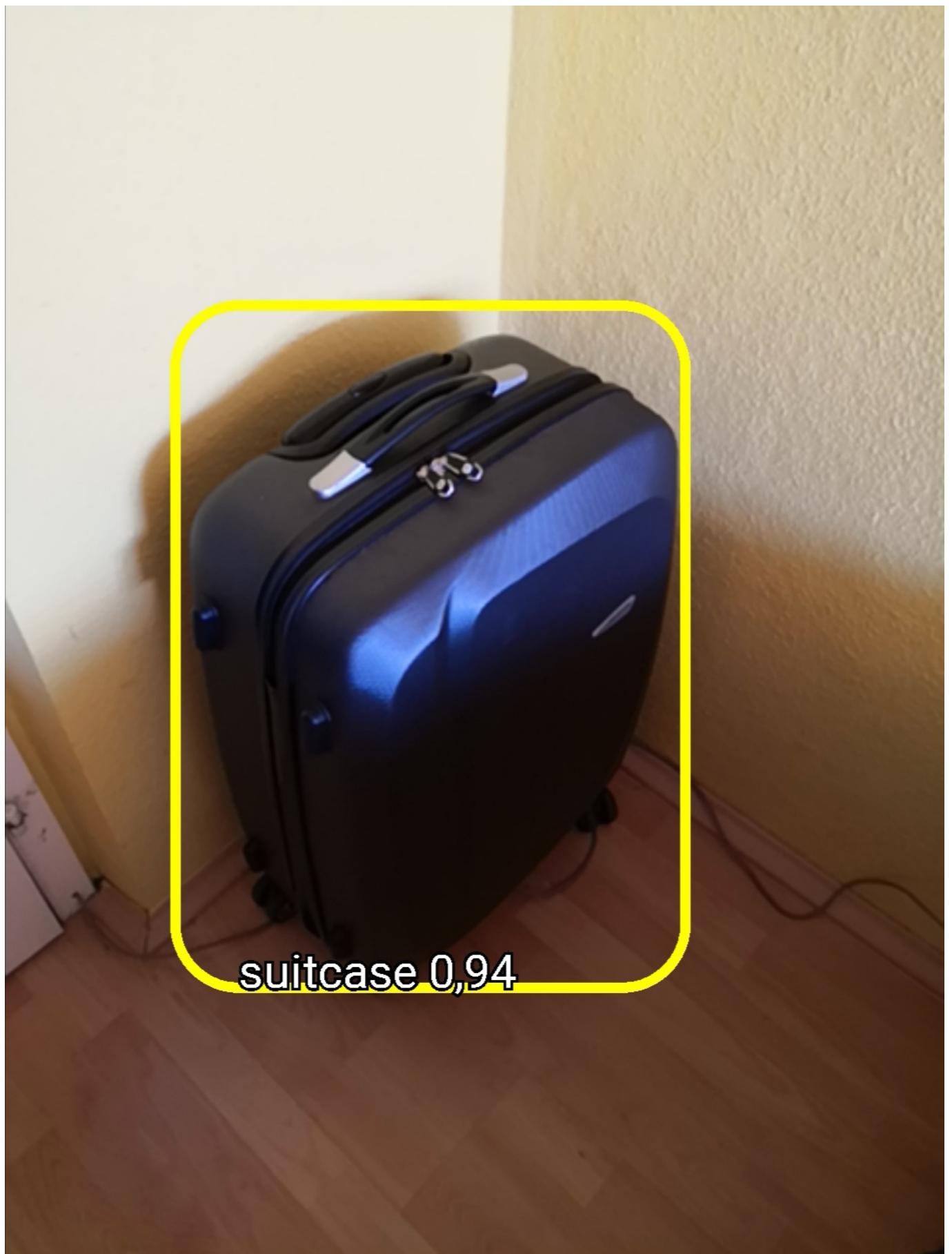
CIFAR-10

Android telefonra a TF objektum detektálója

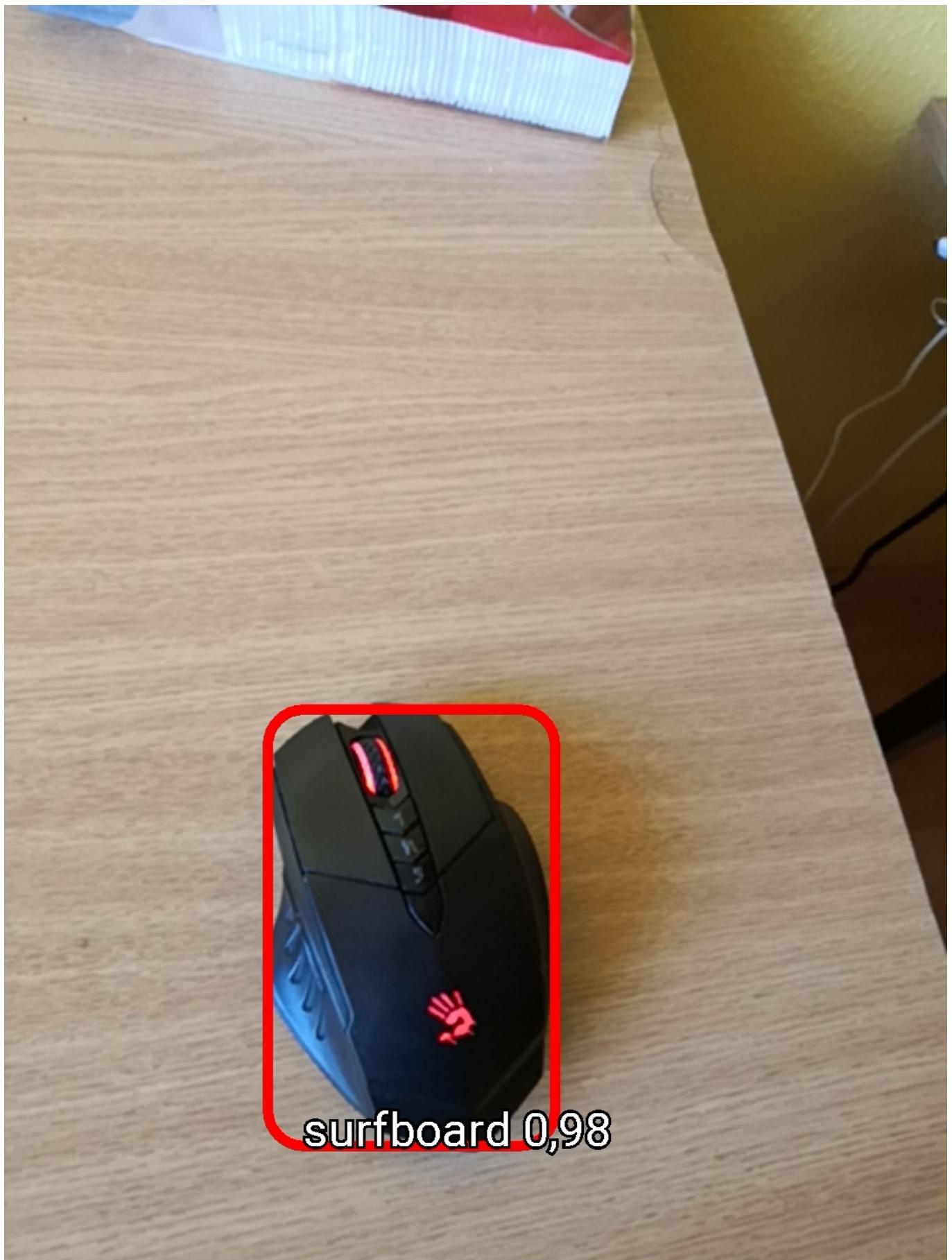
A feladat az volt, hogy próbáljuk ki a tensorflow objektum detektálóját androidon. Ez a program már elérhető Google Play-en így nem volt szükség arra, hogy githubról klónozzam és onnan telepítsem androidra. A kipróbálás már egyszerű csak ráírányítjuk a tárgyra amit szeretnénk, hogy felismerjen és már megy is.

Itt probléma nélkül feismeri az egeret és a bőröndöt:





A program azonban nem mindenkor pontos a hasonló alakú tárgyakat gyakran képes összekeverni.



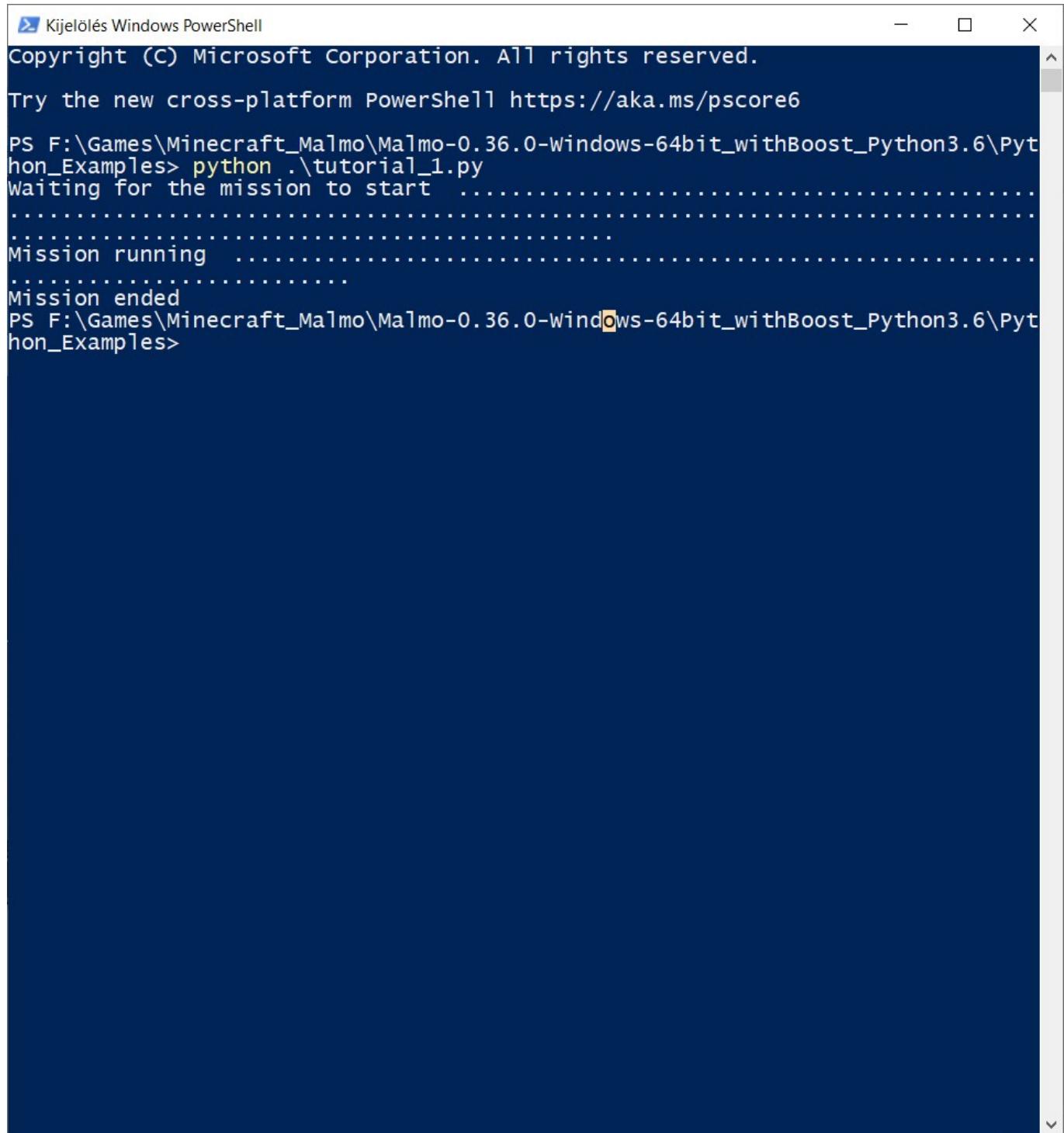
SMNIST for Machines

Minecraft MALMO-s példa

A feladatban életre kellett hívni a Minecraft Malmö-s példát. Ahhoz, hogy ezt megtehessük szükség volt pár segédprogramra. Először is le kellett tölteni a Minecraft Malmöt magát, majd fel kellett telepíteni a Python-t, az 1.8-as OpenJDK-t, a CodeSynthesis XSD-t és az FFmpeg-t. Miután a segédprogramokat telepítettük a környezeti változókba be kellett illeszteni az útvonalakat. Legvégül el kellett indítani a Minecraft\launchClient.bat fájlt, ez lebuildeli a szükséges dolgokat és elindítja a játékot a Malmö moddal.

```
Windows PowerShell
[10:19:47] [Client thread/INFO] [FML]: Itemstack injection complete
[10:19:47] [Forge Version Check/INFO] [ForgeVersionCheck]: [forge] Found status: OUTDATED Target: 13.20.1.2386
[10:19:50] [Sound Library Loader/INFO]: Starting up SoundSystem...
[10:19:50] [Thread-8/INFO]: Initializing LWJGL OpenAL
[10:19:50] [Thread-8/INFO]: (The LWJGL binding of OpenAL. For more information, see http://www.lwjgl.org)
[10:19:51] [Thread-8/INFO]: OpenAL initialized.
[10:19:51] [Sound Library Loader/INFO]: Sound engine started
[10:19:55] [Client thread/INFO] [FML]: Max texture size: 16384
[10:19:55] [Client thread/INFO]: Created: 16x16 textures-atlas
[10:19:56] [Client thread/INFO]: [STDOUT]: CLIENT request state: WAITING_FOR_MOD_READY
[10:19:56] [Client thread/INFO] [FML]: Injecting itemstacks
[10:19:56] [Client thread/INFO] [FML]: Itemstack injection complete
[10:19:56] [Client thread/INFO] [FML]: Forge Mod Loader has successfully loaded 5 mods
[10:19:56] [Client thread/INFO]: Reloading ResourceManager: Default, FMLFileResourcePack:Forge Mod Loader, FMLFileResourcePack:Minecraft Forge, FMLFileResourcePack:Microsoft Malmo Mod
[10:19:57] [Client thread/WARN]: ResourcePack: ignored non-lowercase namespace : MalmoMod in F:\Games\Minecraft_Malmo\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Minecraft\build\libs\MalmoMod-0.36.0.jar
[10:19:57] [Client thread/WARN]: ResourcePack: ignored non-lowercase namespace : MalmoMod in F:\Games\Minecraft_Malmo\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Minecraft\build\libs\MalmoMod-0.36.0.jar
[10:19:57] [Client thread/WARN]: ResourcePack: ignored non-lowercase namespace : MalmoMod in F:\Games\Minecraft_Malmo\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Minecraft\build\libs\MalmoMod-0.36.0.jar
[10:19:59] [Client thread/INFO]: SoundSystem shutting down...
[10:19:59] [Client thread/WARN]: Author: Paul Lamb, www.paulscode.com
[10:19:59] [Sound Library Loader/INFO]: Starting up SoundSystem...
[10:20:00] [Thread-10/INFO]: Initializing LWJGL OpenAL
[10:20:00] [Thread-10/INFO]: (The LWJGL binding of OpenAL. For more information, see http://www.lwjgl.org)
[10:20:00] [Thread-10/INFO]: OpenAL initialized.
[10:20:00] [Sound Library Loader/INFO]: Sound engine started
[10:20:04] [Client thread/INFO] [FML]: Max texture size: 16384
[10:20:04] [Client thread/INFO]: Created: 512x512 textures-atlas
[10:20:06] [Client thread/WARN]: Skipping bad option: lastServer:
[10:20:07] [Client thread/INFO]: [STDOUT]: CLIENT enter state: WAITING_FOR_MOD_READY
[10:20:07] [Thread-12/INFO]: [STDOUT]: INFO: ->mcp(0) Listening for messages on port 10000
[10:20:07] [Client thread/INFO]: [STDOUT]: CLIENT request state: DORMANT
[10:20:07] [Client thread/INFO]: [STDOUT]: CLIENT enter state: DORMANT
[10:20:08] [Realms Notification Availability checker #1/INFO]: Could not authorize you against Realms server: Invalid session id
> Building 95% > :runClient
```

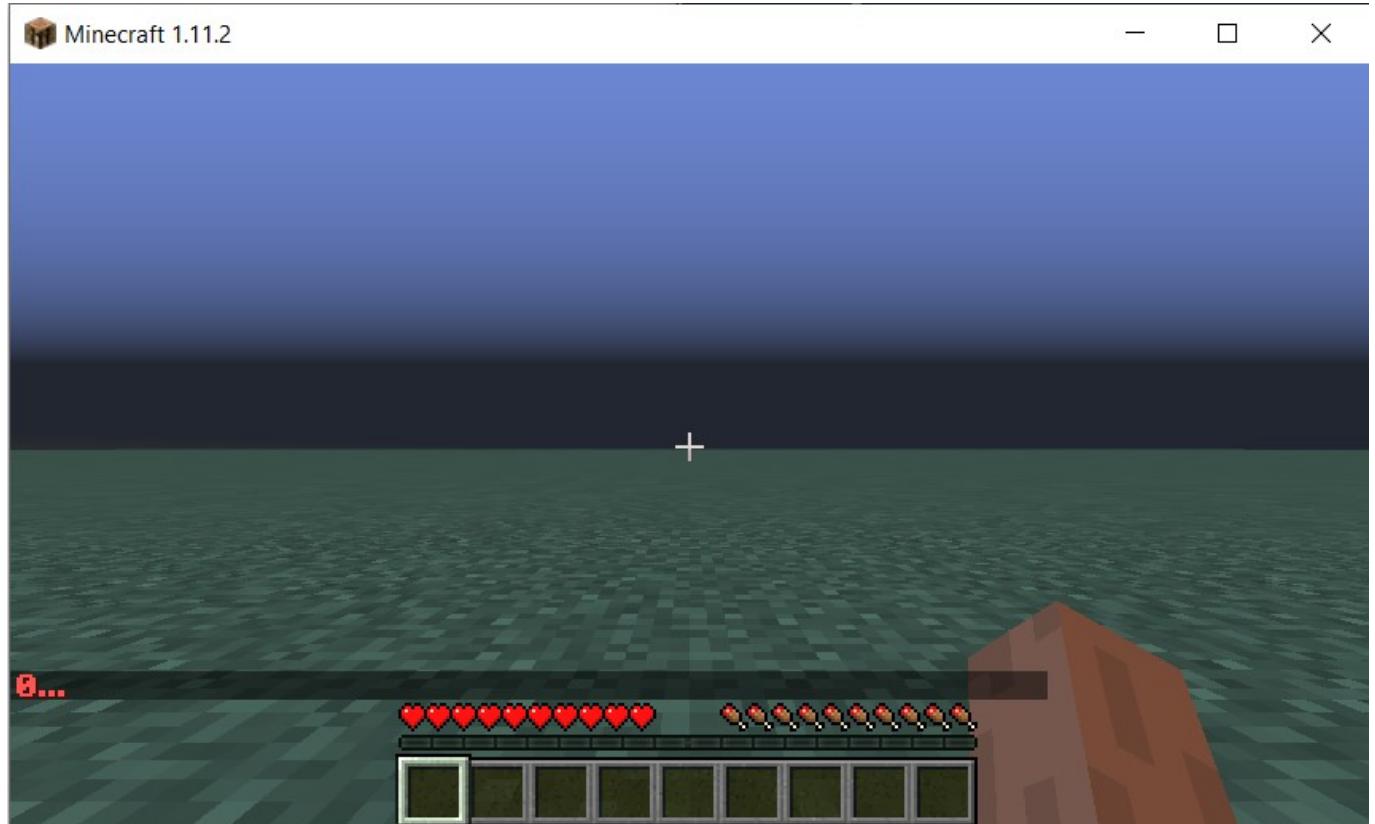
Miután fut a Minecraft egy másik konzolból tudjuk indítani a Python példákat, úgynevezett küldetéseket.
pl:python .\tutorial_1.py A legelő küldetésben még nem történik semmi csak visszaszámol 10-től egy számláló.



```
Kijelölés Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Games\Minecraft_Malmo\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Python_Examples> python .\tutorial_1.py
Waiting for the mission to start ..... .
.
.
.
Mission running ..... .
.
.
.
Mission ended
PS F:\Games\Minecraft_Malmo\Malmo-0.36.0-Windows-64bit_withBoost_Python3.6\Python_Examples>
```



III. rész

Irodalomjegyzék

Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEAHCoders> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.