

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Dr. Bátfai Norbert

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>	<i>SIGNATURE</i>
WRITTEN BY	Szűcs, Levente	2019. november 20.	

**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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

## 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]

# Tartalomjegyzék

<b>I. Második felvonás</b>	<b>1</b>
<b>1. Helló, Arroway!</b>	<b>3</b>
1.1. Olvasónapló	3
1.2. OO szemlélet	3
1.3. Gagy	9
1.4. Yoda	11
<b>2. Helló, Liskov!</b>	<b>13</b>
2.1. Liskov helyettesítés sértése	13
2.2. Szülő-gyerek	14
2.3. Anti OO	16
2.4. Ciklomatikus komplexitás	17
<b>3. Helló, Mandelbrot!</b>	<b>18</b>
3.1. Reverse engineering UML osztálydiagram	18
3.2. Forward engineering UML osztálydiagram	19
3.3. BPMN	20
3.4. BPEL Helló, Világ! egy visszhang folyamat	21
<b>4. Helló Chomsky!</b>	<b>23</b>
4.1. Encoding	23
4.2. Full screen	26
4.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	29
<b>5. Helló, Stroustrup</b>	<b>32</b>
5.1. JDK osztályok	32
5.2. Másoló-mozgató szemantika	36
5.3. Változó argumentum ctor	36

<b>6. Helló, Gödel</b>	<b>40</b>
6.1. Gengszterek	40
6.2. STL map érték szerinti rendezése	41
6.3. Alternatív Tabella	43
6.4. GIMP Scheme hack	44
<b>7. Helló, Valaki!</b>	<b>47</b>
7.1. FUTURE tevékenység editor	47
7.2. SamuCam	50
7.3. BrainB	53
<b>8. Helló, Gödel</b>	<b>55</b>
8.1. Port Scan	55
8.2. Android Játék	56
8.3. Junit Test	61
<b>9. Helló, Berners-Lee!</b>	<b>65</b>
9.1. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I--II.II.	65
9.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobil-programozásba. I--II.II.	65
<b>II. Irodalomjegyzék</b>	<b>67</b>
9.3. Általános	68
9.4. C	68
9.5. C++	68
9.6. Lisp	68

# Táblázatok jegyzéke

2.1. Összehasonlítás . . . . .	17
--------------------------------	----

# 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 alkálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Mindenesetre 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 gyerekeknek, 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 gyerekeknek, 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 mást is) példával.

## Hogyan nyomjuk?

Ránts 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 dlatex 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
dlatex 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 'dlatex' 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 találsz az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

# **I. 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.

---

DRAFT

---

# 1. fejezet

## Helló, Arroway!

### 1.1. Olvasónapló

Az objektumorientált programozás lényegét az adatabsztrakció, öröklődés és a polimorfizmus szavakkal foglалhatjuk össze. Az öröklődés legegyszerűbb esete amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk.

A kiterjesztés jelentheti új műveletek és/vagy új változók bevezetését.

Az osztályok rokonsági viszonyainak összességét osztályhierarchiának hívjuk amit gyakran mint fentről lefelé nöő fát ábrázolnak az Object osztályból ágaztatva.

Az Object osztály egy kiemelt osztály amely java.langba csomagba tartozik. Object implicit módon minden osztálynak szülője amely nem más osztályból származik vagyis amelynek definíciójában nem adjuk meg az extends tagot. Object minden más osztálynak az őse. Az object típusú változók felelnek meg a Java nyelvben a típus nélküli mutatóknak, hiszen tetszőleges típusú objektumra (beleértve tömböket is) hivatkozhatnak.

Minden elemre külön lehet szabályozni annak láthatóságát. Public a külvilág számára látható. Egy elem lehet csak a leszármazottak(protected) vagy senki más számára sem látható(private) is. Ha a megjelölés elmarad, akkor az adott elem csak a forrásszöveg a környezetében, az adott csomagban lesz látható, a külvilág számára nem.

A nyelvnek van egy másik fontos eleme, amely nem része az osztályoznak, ez az interfész. Az interfész már az Objective-C nyelvben is szerepelt protokoll néven, ezt a Java fejlesztői is átvették és kissé módosították. Egy interfész egy új referencia típus, absztrakt metódusok deklarációjának és konstans értékeknek az összessége. Az interfészek az egyik alapvető tulajdonsága, hogy benne található metódusok csak deklarálódnak, vagyis megvalósítás nélkül szerepelnek.

Az interfészek felületet definiálnak. Egy interfész tényleges használata az implemetációján keresztül történik. Egy osztály implementál egy interfészt, ha az összes interfész által specifikált metódushoz implemetációt ad. Ezáltal az absztrakt program konkréttá válik.

### 1.2. OO szemlélet

A legelső feladatunk amellyel foglalkoznunk kellett az a Polárgenerátor. Ezzel a Polárgenerátorral random számokat tudunk képezni. Először Javában majd C++ nyelven kellett megnézni ezeket a feladatokat a fólia jegyzeteinek köszönhetően, nem volt nehéz dolgunk.

A kód :

```
>public class PolarGenerator{

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator(){
        nincsTarolt = true;
    }

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

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

                w = v1 * v1 + v2 * v2;

            } while(w >1);

            double r = Math.sqrt((-2 * Math.log(w)) / w);

            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;
            return r * v1;

        } else{
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

    public static void main(String[] args){

        PolarGenerator g = new PolarGenerator();

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

Vegyük is sorjába a dolgokat.

```
public class PolarGenerator{
```

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

Ebbe az osztályunkba 2 változót fog tartalmazni. Amint láthatjuk, hogy a 2 változó típusa különböző, először is a nincsTarolt egy boolean típusú változó amelynek értéke vagy igaz vagy hamis lehet. Ez a változóba mondja meg, hogy van-e tárolt változónk vagy nincsen. A másik változóba a taroltba pedig a változó értékét adjuk meg tizedes törtben.

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

Ezután ebbe konstruktorba a nincsTarolt-at igazgá tesszük.

```
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;
    }
}
```

Ebbe a kovetkezo nevű függvény fut le a számítás. Ha nincs tárolt értékünk akkor 2 értéket számolunk. Akkor ha van tárolt értékünk, akkor pedig azt a tárolt értéket visszaadjuk.

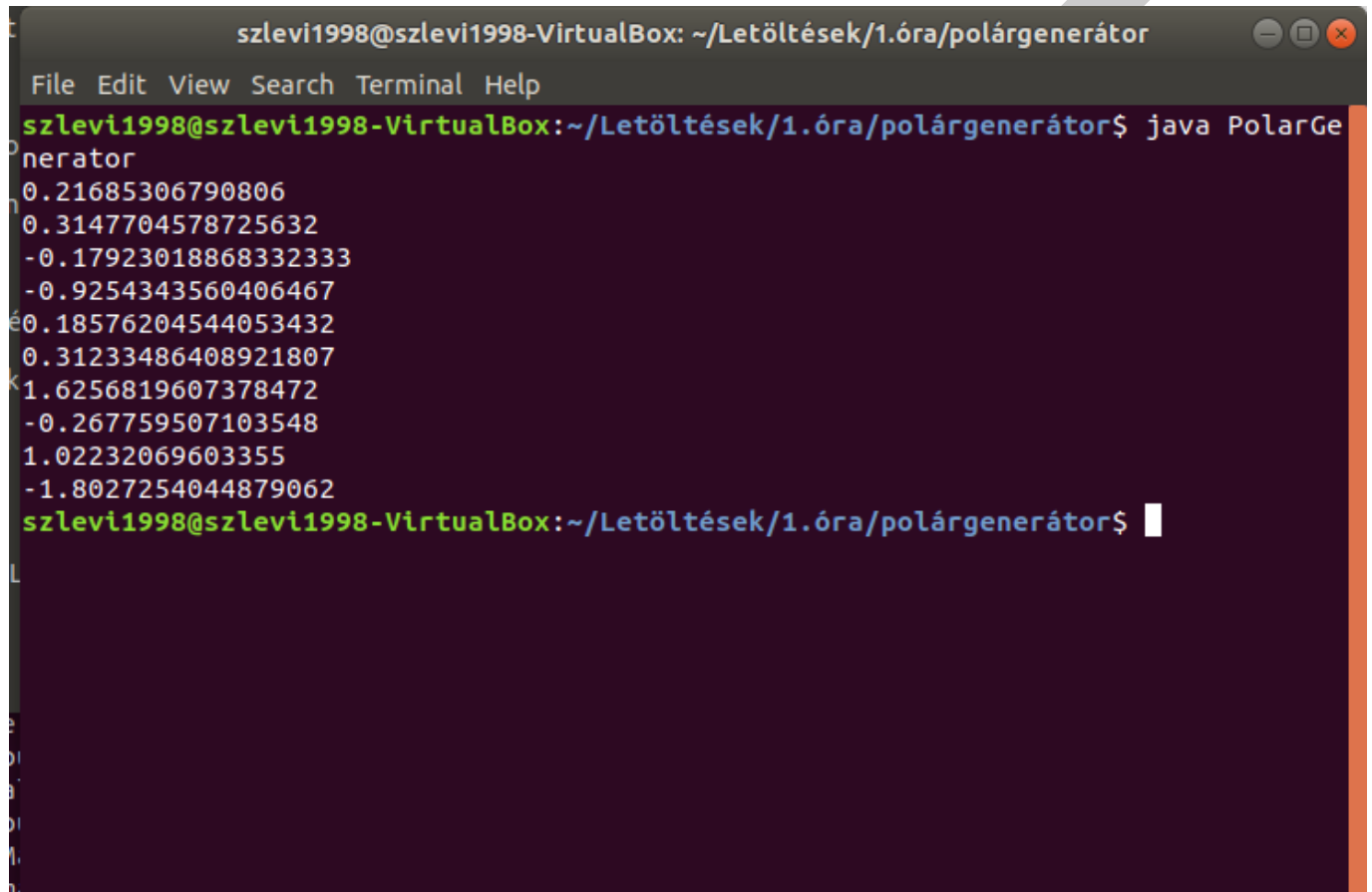
```
public static void main(String[] args) {

    PolarGenerator g = new PolarGenerator();

    for(int i = 0; i < 10; ++i)
```

```
System.out.println(g.kovetkezo());  
}
```

Ezután a mainben lefut a programunk. Példányosítjuk a PolárGenerátor osztályt. Ezután egy for ciklusban 10-szer hívjuk meg a kovetkezo() metódust. A program 10 darab random számot fog nekünk kiírni amit itt is láthatunk.



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/polárgenerátor  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ java PolarGe  
nerator  
0.21685306790806  
0.3147704578725632  
-0.17923018868332333  
-0.9254343560406467  
0.18576204544053432  
0.31233486408921807  
1.6256819607378472  
-0.267759507103548  
1.02232069603355  
-1.8027254044879062  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$
```

Ez a program nagyon hasonlít az eredeti, a JDK-s src fájljaiban is megtalálható random generátorra. Itt láthatjuk a Java.util.Random osztályban lévő Randomot és látható benne a hasonlóság.

```
public synchronized double nextGaussian()
{
    if (haveNextNextGaussian)
    {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    }
    double v1, v2, s;
    do
    {
        v1 = 2 * nextDouble() - 1; // Between -1.0 and 1.0.
        v2 = 2 * nextDouble() - 1; // Between -1.0 and 1.0.
        s = v1 * v1 + v2 * v2;
    }
    while (s >= 1);
    double norm = Math.sqrt(-2 * Math.log(s) / s);
    nextNextGaussian = v2 * norm;
    haveNextNextGaussian = true;
    return v1 * norm;
}
```

Ezután a programunkat megírtuk C++ -ban is. A kódok itt vannak:

Először is a header file:

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time (NULL));
    }
    ~PolarGen()
    {
    }
    double kovetkezo();

private:
    bool nincsTarolt;
    double tarolt;
```



```
};  
#endif
```

Itt található a fájlunknak a header fájlja. Ebbe találhatóak meg az osztályok. Két elkülöníthető osztályunk van a public és a private. A public az osztályon kívül is elérhető. Ezzel szembe a private csak az adott osztályon belül érhető el.

A privateba megtalálható ugyan az a két változónk itt is az egyik változó bool a másik pedig szintén egy tört változó csak úgy mint a Javas kódunkban.

```
#include "polargen.h"  
  
double  
PolarGen::kovetkezo ()  
{  
    if (nincsTarolt)  
    {  
        double u1, u2, v1, v2, w;  
        do  
        {  
            u1 = std::rand () / (RAND_MAX + 1.0);  
            u2 = std::rand () / (RAND_MAX + 1.0);  
            v1 = 2 * u1 - 1;  
            v2 = 2 * u2 - 1;  
            w = v1* v1 + v2 * v2;  
        }  
        while(w > 1);  
  
        double r = std::sqrt ((-2 * std::log (w)) / w);  
  
        tarolt = r * v2;  
        nincsTarolt = !nincsTarolt;  
  
        return r * v1;  
    }  
    else  
    {  
        nincsTarolt = !nincsTarolt;  
        return tarolt;  
    }  
}
```


Amint látható a két nyelv között nincsen óriási különbség. Szintén a számítás csak úgy mint a Java-ban a kovetkezo függvényben megy létre. Abban az esetben, ha a nincsTarolt igaz akkor egy random törtszámot generál, ha hamis akkor pedig visszaadja az előzőleg tárolt értéket.

```
int main (int argc, char **argv)  
{  
    PolarGen pg;
```

```
for (int i = 0; i < 10; ++i)
    std::cout << pg.kovetkezo() << std::endl,

return 0;
}
```

Csak úgy mint a Javában, itt is létrehozuk a mainben a pg PolarGen osztályt és ezután, a for ciklus 10-szer lefut, és a kovetkezo() kiíratja a random számokat. Ezeket az értékeket láthatjuk:



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/polárgenerátor
File Edit View Search Terminal Help
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ g++ polargenteszt.cpp -o Polar
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ ./Polar
1.67712
-0.734798
-1.47623
1.02756
-1.03194
-1.07785
-0.256456
0.710424
-0.339193
-0.744762
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$
```

### 1.3. Gagyí

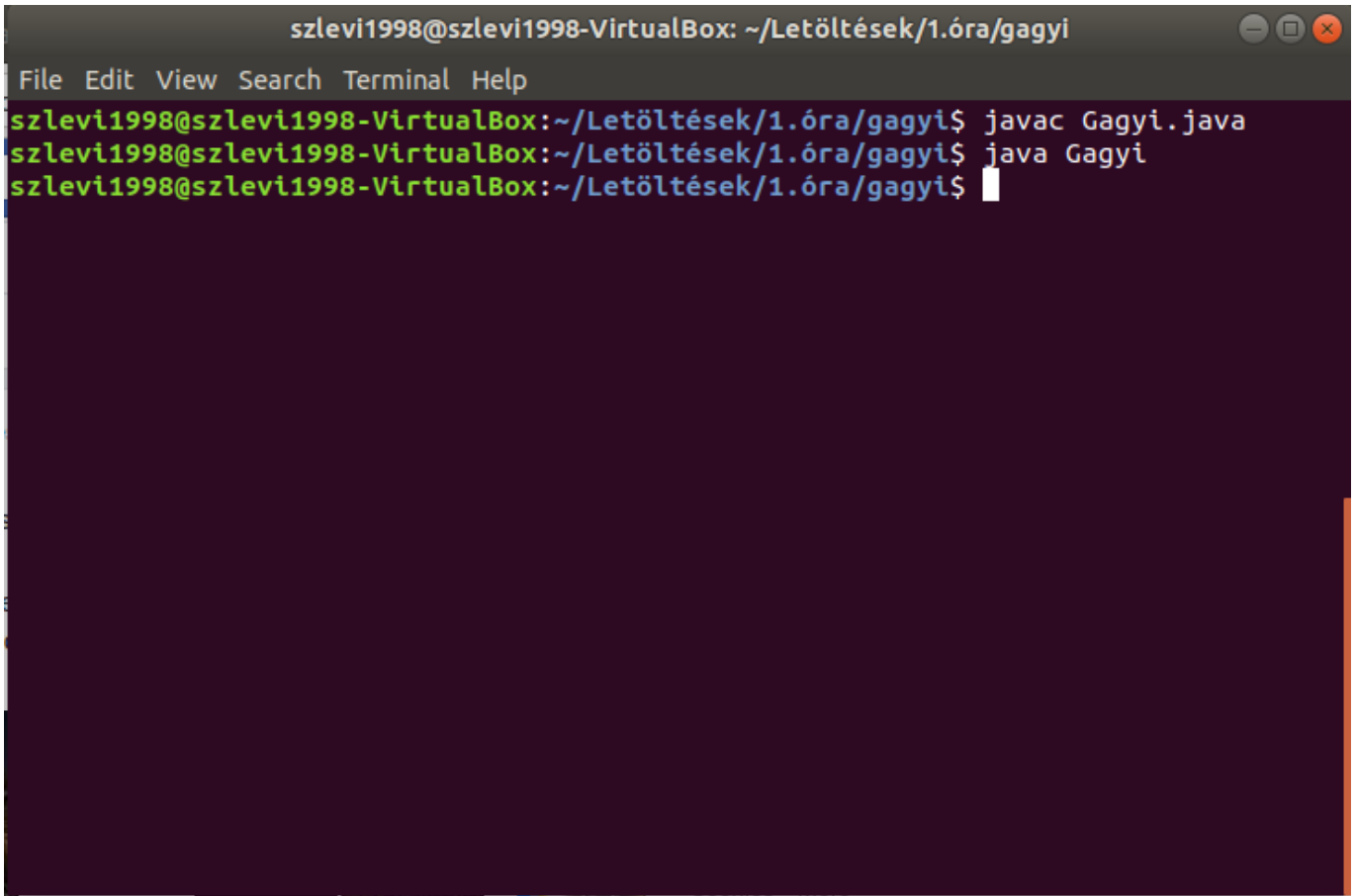
```
while (x <= t && x >= t && tt != x);
```

Ebben a feladatban ezzel a kódcsipettel foglalkoztunk, a feladatunk az volt, hogy ezzel a példaprogrammal egyszer végtelen ciklust, más értékkel pedig ne okozzon.

```
Public class Gagyí
{
    public static void main(String[] args){
        Integer t = -128;
        Integer x = -128;

        while (x <= t && x >= t && tt != x){
            System.out.println("yes");
        }
    }
}
```

Ebben a kódcsipetben két Integer objektumot hozunk létre. Fontos tudni, hogy az Integer objektumok egy előre készített poolból veszi az értékeit -128-tól 127-ig. Ilyenkor két objektumunk a memóriacímük megegyezik, azért mert a poolban lévő értékeknek a memóriacímük megegyezik. Ez a while ciklus hamis lesz, azért mert a != operátor feltétele teljesül, ezért nincs végtelen ciklus.



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/gagyí
File Edit View Search Terminal Help
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$ javac Gagyi.java
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$ java Gagyi
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$
```

Viszont abban az esetben, hogy ha az értékünk -129, akkor ez már nem fog teljesülni hiszen a -129 nem a pool része ezért a memóriacímük nem fog megegyezni, ezért a while ciklus igaz és ez végtelen ciklust hoz létre.

[illegible]

### 1.4. Yoda

Ebben a feladatban a Yoda feltételekkel foglalkozunk. Köztudott, hogy Yoda a Star Wars filmekben nyelvtantunkhoz képest fordítottan beszél, hiszen először az igéket használja, majd a tárgyat mondja. Pl: "I have a phone." Yoda "nyelvtannal" így hangzik : "Phone, I have."

Yoda feltételekben szokástól eltérően a változó értékét bal oldalra írjuk míg jobbra kerül maga a változó neve. Vannak olyan esetek amelyben hasznos tud lenni, azonban nem szokták legtöbbször alkalmazni. Hasznos tud lenni ha null pointer Exception hibákat akarunk elkerülni, azonban a veszélye így is fennáll, hogy ez a hiba későbbiekben fennáll. Ami előny, hogy akkor amikor a konstans bal oldalra kerül, akkor alapvetően nem változtatja meg a program működését. Hátrányok közé sorolható, mint említettem, hogy a null pointer Exception hibákat csak elrejt, hiba továbbra is fennállhat, illetve olvashatóság szempontjából sem a legideálisabb.

Megszokott szintaxis:

```
int yoda = 5;
if (yoda == 5){
System.out.println("this statement is true");
}
```

## Yodás szintaxis

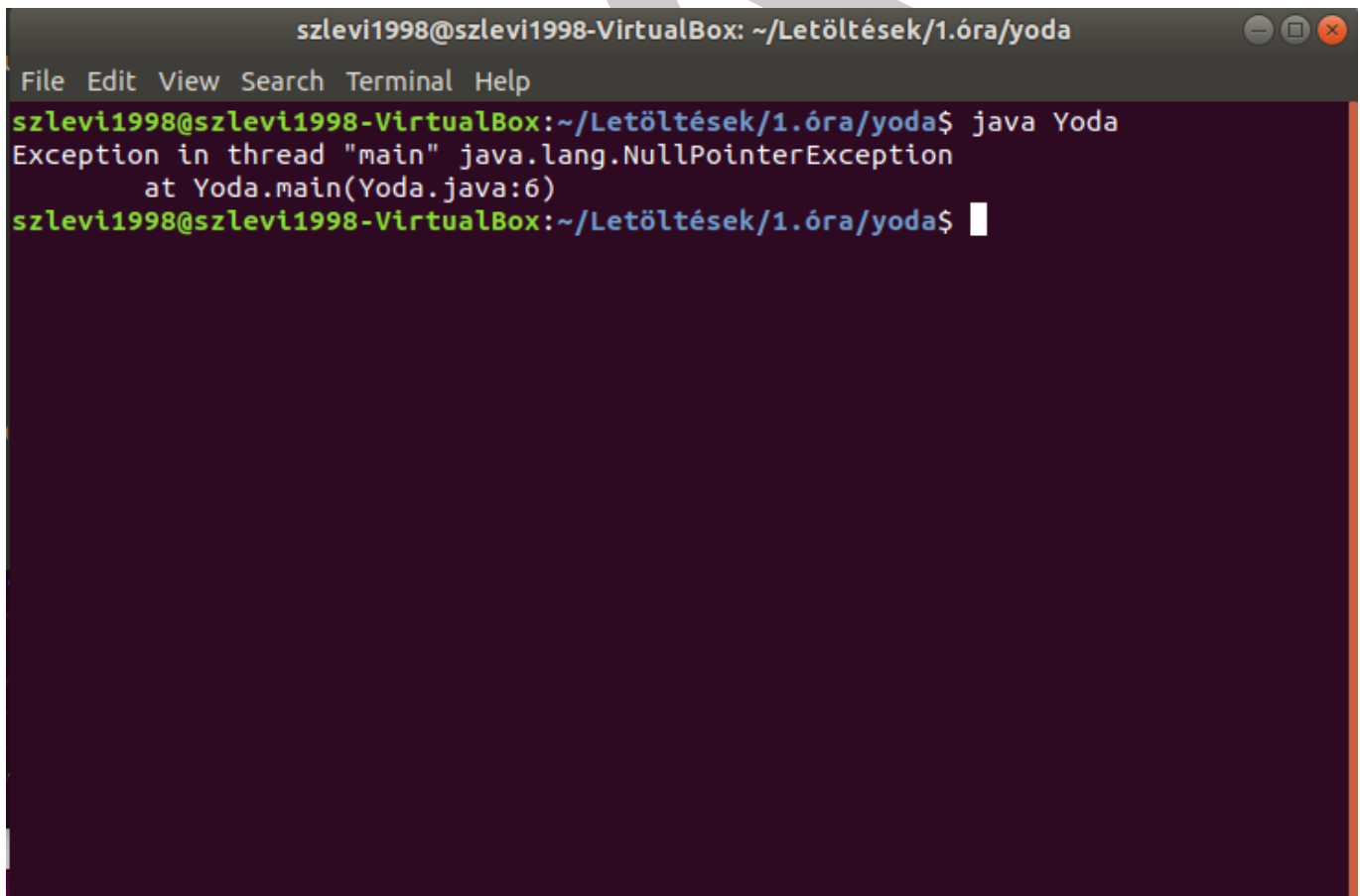
```
int yoda = 6;
if (6 == yoda) {
    System.out.println("This statement is also true");
}
```

Amint láthatjuk hogy az érték és a változó megcserélődött ebbe a szintaxisban.

És akkor a kód:

```
public class Yoda
{
    public static void main(String[] args)
    {
        String Yoda = null;
        if(Yoda.equals("something")) {
            System.out.println("valami más");
        }
    }
}
```

A kód nem követi a Yoda conditions és ezért a programunk leáll java.lang.nullPointerExceptionnel.

A screenshot of a terminal window titled "szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/yoda". The terminal shows the command "java Yoda" being executed, which results in a "NullPointerException" error. The error message is displayed in red text: "Exception in thread "main" java.lang.NullPointerException at Yoda.main(Yoda.java:6)". The prompt "szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/yoda\$" is visible at the bottom of the terminal window.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/yoda
File Edit View Search Terminal Help
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/yoda$ java Yoda
Exception in thread "main" java.lang.NullPointerException
    at Yoda.main(Yoda.java:6)
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/yoda$
```

## 2. fejezet

# Helló, Liskov!

### 2.1. Liskov helyettesítés sértése

Ebben a feladatban a Liskov helyettesítés sértésére kellett programot írunk. A híres S.O.L.I.D programozásnak (Tiszta kódolás) az L betűje a Liskov-elv amely arról szól, hogy minden osztályt lehet helyettesíteni a leszármazott osztályával, amellyel a program alapvető működése nem változik. Ebben a feladatban segítségül kaptuk az Udprog repójában lévő forrást. Ez alapján csináltam egy saját programot. Íme :

```
class Jarmuvek{
public:
    virtual void gurul(){};
};

class Program {
public:

    void fgv ( Jarmuvek &jarmuvek ) {
        jarmuvek.gurul();
    }
};

class Car : public Jarmuvek
{};

class Hajo : public Jarmuvek
{};

int main ( int argc, char **argv )
{
    Program program;
    Jarmuvek jarmuvek;
    program.fgv ( jarmuvek);

    Car car;
    program.fgv ( car );
}
```

```
Hajo hajo;  
program.fgv ( hajo );  
  
}
```

Ebben a kódban a Járművek osztály az űsosztály és ebben a gyermekosztályok az autók és a hajók. Itt van egy gurul függvény, amely a problémát okozza, itt minden egyes gyerekosztályra utal azaz a Carra és a hajóra. A Carral semmi gond azonban, a hajó nem tud gurulni, viszont ez így nem igaz ezért a Liskov-elv sérül.

## 2.2. Szülő-gyerek

Ebben a feladatban a szülő és gyerek osztály kapcsolatáról foglalkozunk. Ez a feladat a polimorfizmussal és leginkább az öröklődéssel foglalkozik.

Az objektumorientált programozás lényegét az adatabsztrakció, öröklődés és a polimorfizmus szavakkal foglалhatjuk össze. Az öröklődés legegyszerűbb esete amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk.

C++ és Javaban kellett egy egyszerű programot írunk, ahol azt kellett demonstrálni, hogy a szülő osztályon csak a szülő üzeneteit lehet küldeni, a gyerek osztálynak az elemeit pedig nem.

Itt látható a kód C++-ban :

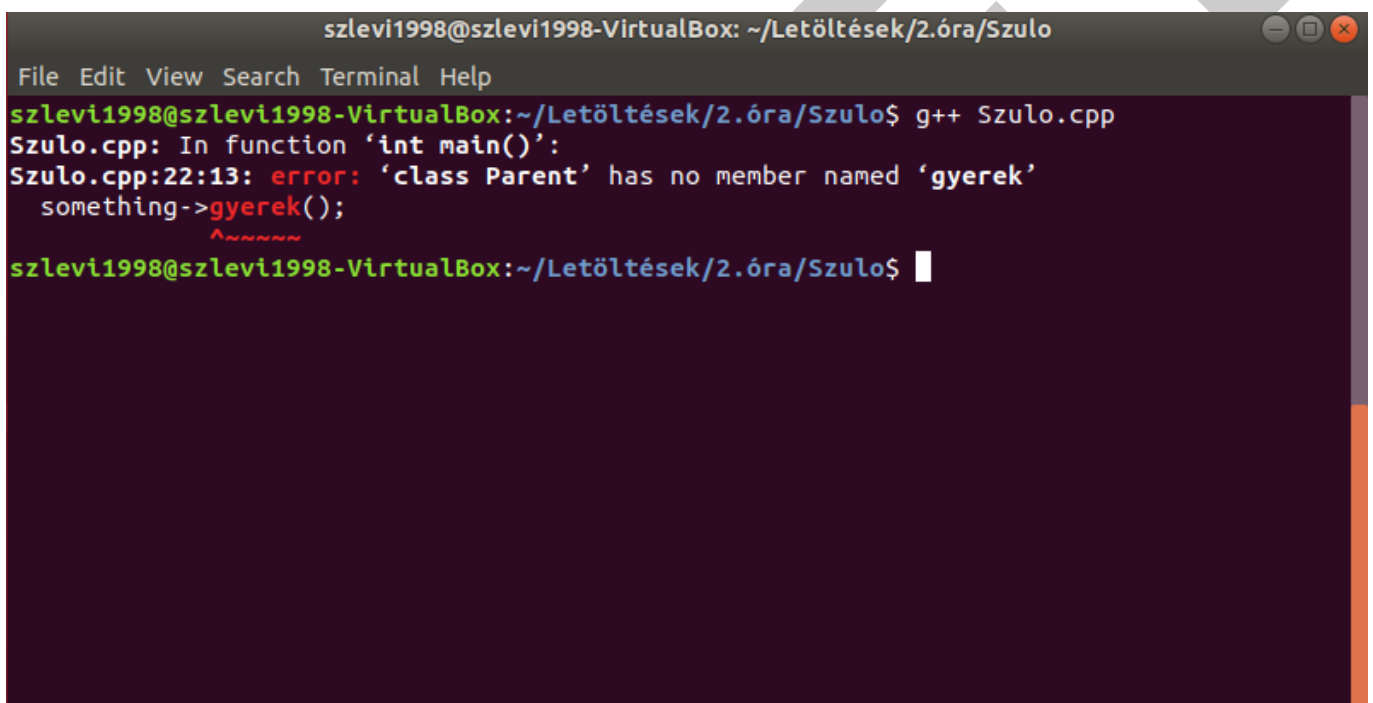
```
    ]  
#include <iostream>  
using namespace std;  
  
class Parent {  
public :  
    void parent() {  
        std::cout << " Ez a szülő osztály " << std::endl;  
    }  
};  
  
class Gyerek : public Parent {  
public :  
    void gyerek() {  
        std::cout << " Ez a gyerek osztály " << std::endl;  
    }  
};  
  
int main(){  
    Parent* something = new Gyerek();  
    something->parent();  
    something->gyerek();  
}
```

Amint látható a kódon van két osztályunk a Parent és a Gyerek osztály.

```
class Gyerek : public Parent {
```

Ebben a sorban látható az, hogy a Gyerek osztálynak átadjuk a Szülő osztályfunkcióit. A c++-ban így lehet átadni egy gyerek osztálynak a szülő osztály funkcióit.

Ezután a mainben példányosítunk és meghívjuk ezeket a függvényeket. A gyerek tudja használni a saját és a szülő osztály funkciót, azonban a szülő csak a saját funkcióit tudja alkalmazni, a gyereket viszont nem. Amint látható a program emiatt a hiba miatt nem is tud lefordulni.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'g++ Szulo.cpp' being executed. The output indicates an error in 'Szulo.cpp' at line 22, column 13: 'error: 'class Parent' has no member named 'gyerek''. The error message points to a call to 'something->gyerek();'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$ g++ Szulo.cpp
Szulo.cpp: In function 'int main()':
Szulo.cpp:22:13: error: 'class Parent' has no member named 'gyerek'
  something->gyerek();
             ^~~~~~
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$
```

Itt van a kód Javában is :

```
class Parent {

    void parent() {
        System.out.println("Ez a szülő osztály");
    }
}

class Gyerek extends Parent {
    void gyerek() {
        System.out.println("Ez a gyerek osztály");
    }
}
```

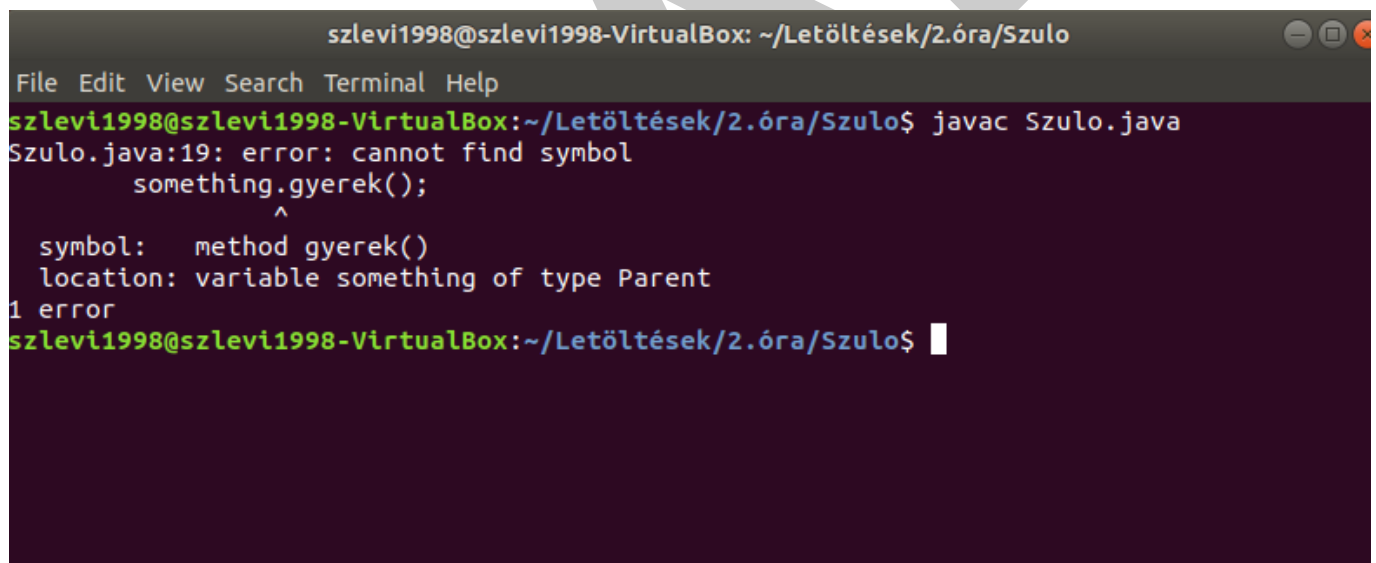


```
class Szulo {  
    public static void main(String[] args){  
        Parent something = new Gyerek();  
  
        something.parent();  
        something.gyerek();  
    }  
}
```

Amint látható egy nagyobb különbség található meg a C++ és a Java közötti programban. A két nyelv szintaktikája alapvetően nagyon hasonlít egymásra. Azonban az öröklődés egy kicsit különbözik.

```
class Gyerek extends Parent
```

Amint látható az öröklődést itt az extends kulcsszóval lehet alkalmazni. Amint látható, nincs eltérés, hiszen a gyerek képes alkalmazni a saját és a szülő osztály funkciót, és itt is a szülő csak a saját funkciót tudja elérni, a gyereket nem.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'javac Szulo.java' being executed. The output is an error message: 'Szulo.java:19: error: cannot find symbol something.gyerek();' with an arrow pointing to the 'gyerek()' part. Below this, it says 'symbol: method gyerek()' and 'location: variable something of type Parent'. At the bottom, it says '1 error'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$ javac Szulo.java  
Szulo.java:19: error: cannot find symbol  
    something.gyerek();  
                ^  
    symbol:   method gyerek()  
    location: variable something of type Parent  
1 error  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$
```

## 2.3. Anti OO

Ebben a feladatban a BBP algoritmus futási időit hasonlítgatjuk össze. Sajnos az erősebb gépen a Linux nagyon sokat crashel ezért nem tudok teljes eredményt mutatni, ezért csak virtualboxos adatot tudok felmutatni. Az alábbi eredményeket kaptam:

Amint látható a Java a leggyorsabb nyelv míg a C a leglassabb. A C volt a leglassabb, ami annyira nem is meglepő, hiszen ez a nyelv a legelavultabb és mivel ez a legrégebbi, ezért nem meglepetés, az sem, hogy a legkevésbé optimalizáltabb. A Java nyert, hiszen ennek a legjobb a memória menedzselése. Érdekes volt az látni, hogy ahogy növekedett a hatvány értéke, úgy növekedett az űr a teljesítmények között.

	C	C#	Java
10 <sup>6</sup>	4.168	3.788	3.614
10 <sup>7</sup>	43.736	44.873	41.073
10 <sup>8</sup>	509.181	487.927	456.563

## 2.1. táblázat. Összehasonlítás

## 2.4. Ciklomatikus komplexitás

Ciklomatikus komplexitás egy olyan szoftveres "mértékegység" amellyel a programunknak a komplexitását tudjuk számmal leírni. Ez a mérés a gráfelmélet alapján történik. Ezt a fogalmat McCabe komplexitásnak is nevezzük. Itt a képlete:

$$M = E - N + 2P$$

M az a szám amely a komplexitást adja meg. E az a szám amely az éleknek a számát adja abból kivonjuk a gráfnak csúcsai és hozzáadjuk az összes komponens dupláját. Én a ennek a dián([https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_2.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf)) 78. oldalán lévő programkódcsipetnek a komplexitását mértem. A [www.lizard.ws](http://www.lizard.ws) oldalon néztem meg az eredményét és a következő látható :

Try Lizard in Your Browser

.java
Analyse

```

@Override
protected void jatekbanVezertes(){

    if (latomAKozepet && distaceKozep < 10.0) {
        sarkonFordul();
    } else if (latomASajatGolvonalat && distanceSajatGolvonai < 0.5) {
        sarkonFordul();
    } else if (latomASzelet && distanceSzele < 5.5) {
        palyanMaradni();
    } else if (latomAFocit){
        if (distanceFoci < 0.7) {

```

Code analyzed successfully.

File Type .java
Token Count 190
NLOC 30

Function Name	NLOC	Complexity	Token #	Parameter #
jatekbanVezeries	29	15	186	

Amint láthatjuk a komplexitás ennek a kód részletnek 15. Egy jól megírt, struktúrált kód körülbelül 1 és 10 között van. 15 már egy komplex kódnak számítható, azonban nem olyan komplikált. Ez a kód még jól tesztelhető, azonban ha 20 fölötti az értékünk akkor már egy nagyon komplikált kódról beszélhetünk. 40 fölötti érték már tesztelhetlen és ilyenkor át kell gondolnunk újra a programot hiszen a kódunk túlságosan komplex.

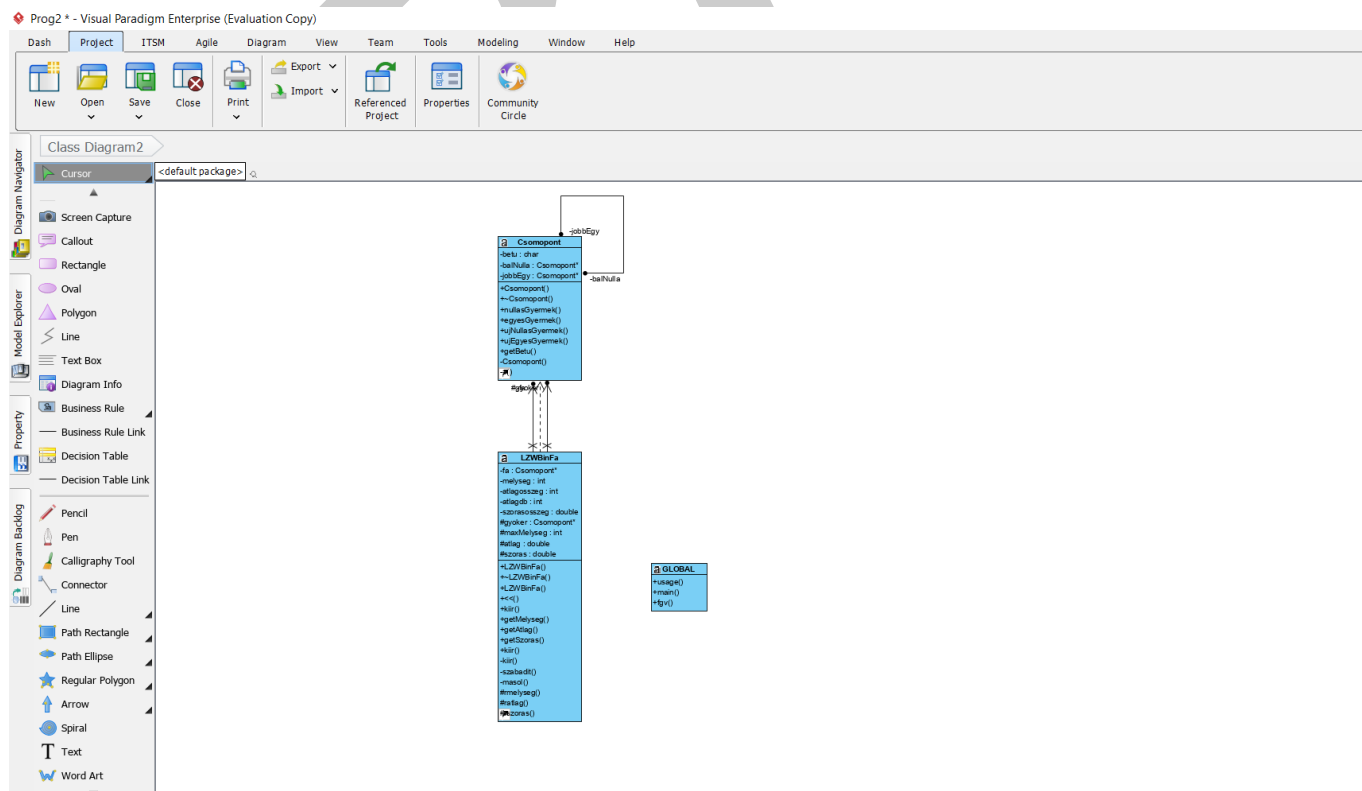
## 3. fejezet

# Helló, Mandelbrot!

### 3.1. Reverse engineering UML osztálydiagram

Az UML egy szabványos, egységesített grafikus modellezőnyelv. A forráskód olyan dokumentum, ami egyértelműen definiálja a program működését. Az UML szabványos lehetőségeket kínál a rendszer felvázolásához, beleértve a fogalmi dolgokat, mint üzleti modellezés és rendszerfunkciók, valamint a konkrét dolgokat, mint programnyelvi utasítások, adatbázis sémák és újrafelhasználható szoftverkomponensek.

Ebben a feladatban az LZW binfájából kellett egy UML osztálydiagramot rajzolni. BPMN feladathoz hasonlóan itt is a visual paradigmát alkalmaztam. Bár nem tudom, hogy melyik Binfa a legmegfelelőbb ehhez a feladathoz, de én a z3a7.cpp-vel próbáltam ki. Elég egyszerű volt a feladat hiszen csak annyit kellett csinálni, hogy a tools opcióba, mint a feladatnevéből kiindulva reverselni kellett a kódot. Ezt az eredményt kaptam :



A Visual paradigm szépen lerajzolja, ezt a binfát UML verzióban. Amint látható jól elkülöníthető részeket láthatunk. Vannak Globális függvényeink, de ezek nem sok vizet zavarnak, hiszen nincsenek ágazásaiak.

Két osztályunk van a Csomópont és a LZWBInfa ezeknek vannak tagjai. Amint láthatjuk vannak jeleink (-+ #) ezek a láthatóságot jelöli. A - jel, azt jelenti, hogy a tagunk private a + a public és a # amely a protected jelzi. Csomópont osztályban van egy olyan tag, amely önmagával kapcsolja össze magát. Ezeket az összekapcsolásokat amely osztályok között vannak asszociációnak nevezzük.

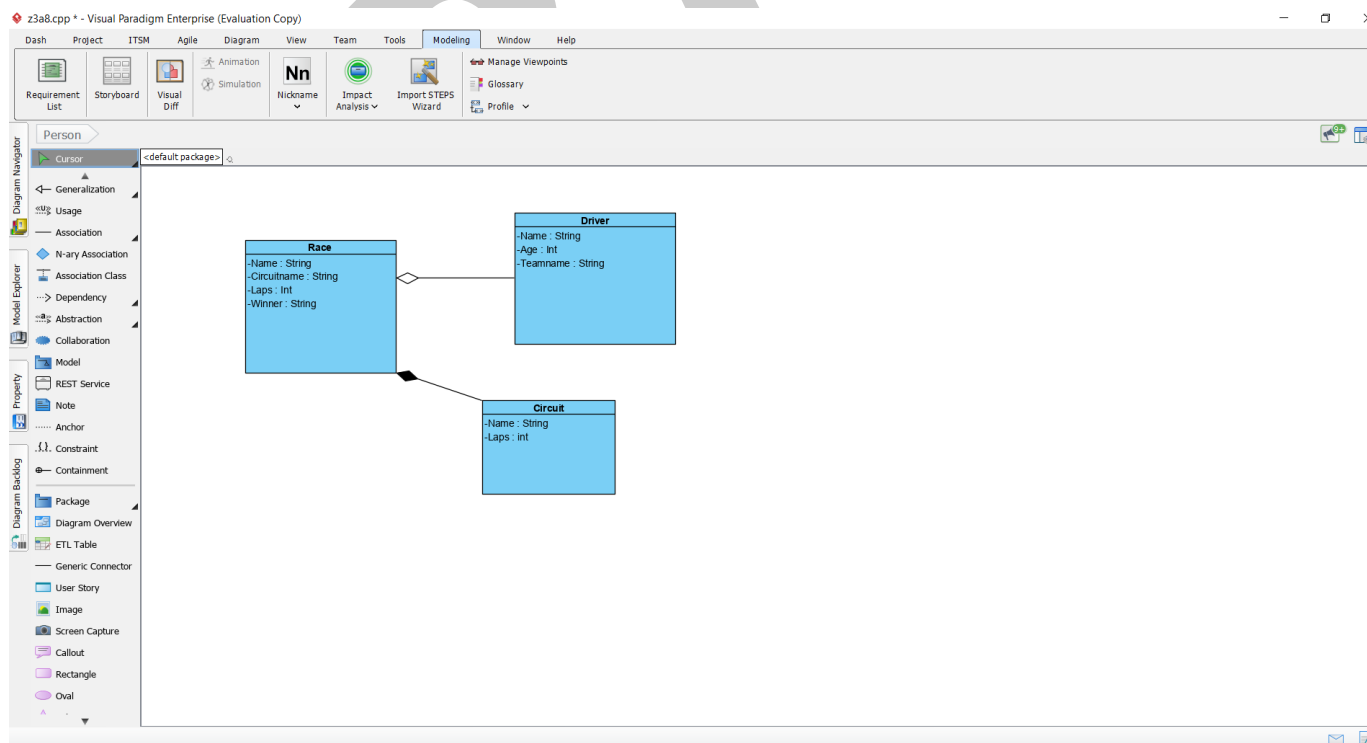
Az asszociációban az osztályok között az osztályok léte független, azonban a osztályok legtöbb esetben legalább egy osztály ismeri a másikat. Az asszociációnak két fajtája van az erős és gyenge aggregáció. A gyenge aggregáció, olyan amikor egy objektum létezhez magában is, de lehet másnak is része. Ennek a jele az UML-ben az üres rombusz.

Az erős aggregációt kompozíciónak is nevezzük, ekkor a részek élettartalma szigorúan megegyezik az egészével. A tartalmazó nem létezhet a tartalmazott nélkül. Ennek jele az UML-ben a jele a telített rombusz.

Bár az én visual paradigmmon ezeket a jelöléseket nem mutatja, mint az erős aggregációt (kompozíciót), viszont ha az egerünket rámutatjuk a kapcsolatra akkor kiírja, hogy asszociáció van a két osztály között. Láthatunk szaggatott vonalas nyilat, amely azt jelenti, hogy dependency (függőség) van az LZWBInfa és a Csomópont osztályok között. Ez azt jelenti, hogy ha valami változás lesz a Csomópont osztályban akkor kihatással lesz az LZWBInfa osztályra is.

## 3.2. Forward engineering UML osztálydiagram

Ebben a feladatban az előző feladatnak az ellentétjét kellett csinálni, itt UML osztályokból kellett programot készíteni. Forward engineering az a lényege, hogy egy program kódot struktúráltan, átláthatóan építsünk fel. Ennek az az előnye, hogyha egy komplikált kódot írunk, akkor gyorsan orvosolhatjuk a problémákat.



Ezen az ábrán van három osztályom, és ebből tudtam kódot generálni a Visual paradigmmeel. Ezzel a programmal és természetesen az UML-s módszerrel nagyon egyszerűen, lehet illusztrálni, hogy hol van

asszociáció, öröklődés a programunkban. Ez azért nagyon praktikus, mert ezzel sokkal gyorsabban lehet változtatni a teljes projekten. Gyakorlatilag egy nyíl "áthúzásával" nagyon sok sornyi programkódot lehet módosítani, átírni. A kisebb projekteken, mint az enyémen, annyira nem hatásos, mert nagyon kevés adatot tartalmaz, de itt is hasznos, mert ellenőrzésnek tökéletesen megfelel.

### 3.3. BPMN

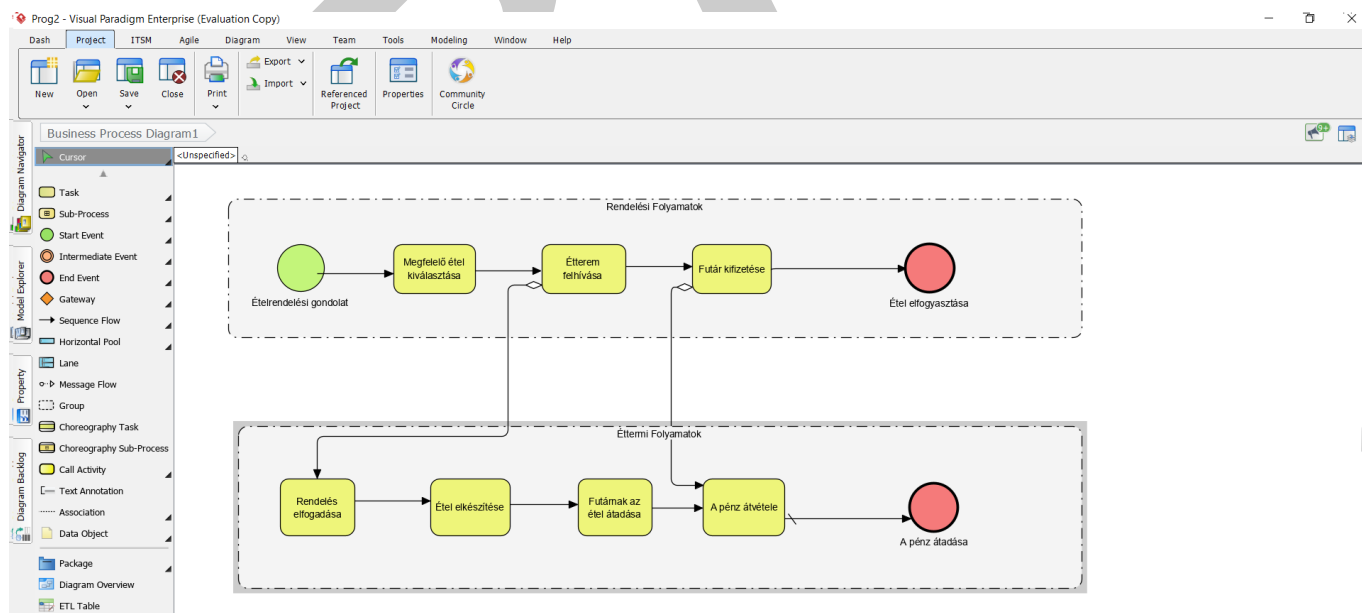
A BPMN (Business Process Modeling Notation) egy olyan folyamatábra, amely az üzleti folyamatok grafikus modellezését szolgálja. Üzleti elemzőknek és technológiai fejlesztőknek szóló grafikus jelölőnyelv.

Az üzleti folyamatok ábrázolására, bemutatására, az EPC és a BPMN módszerek a legelterjedtebbek. Az egyszerű folyamatok ábrázolásától a komplex vállalati folyamatrendszerekig képesek mindent vizualizálni. Ahhoz, hogy a valóságról kapjunk egy képet, amely alapján tudjuk elemezni majd fejleszteni a folyamatainkat elengedhetetlen, hogy azokat grafikailag ábrázoljuk.

Több fajtája is van:

- BPML (XML alapú)
- BPEL (XML alapú)
- XDPL (XML alapú)
- BPMN (Grafikus jelölésre alkalmas)

Ebbe a feladatban rajzolnunk kell egy saját folyamatábrát. Nem igazán alkalmaztam ezelőtt UML-t és ezért volt egy kisebb dilemmám, hogy milyen szoftvert alkalmazzak. Végül a Visual Paradigm-ot választottam, annak is a 30 napos próba verzióját. Egészen egyszerűen lehetett alkalmazni ezt a programot, és elkészítettem a saját folyamatomat. Íme :



Amint a képen látható egy étteremből való rendelését vezettem le. Látható egy a zöld karika, ez a kezdőpontunk. Ez a tevékenység, amely létrehozza, ezt a teljes folyamatot. Láthatunk két nagyobb halmazt, ez a két halmaz, amely a teljes folyamatot képezi. Ez a két halmaz kapcsolatba van egy mással. Láthatjuk, hogy többször is összekapcsoltam a két halmazt. Ez azért van, mert egyes tevékenységet egy adott halmaz nem tehet meg a másik "engedélye nélkül."

Láthatunk a képen sárga téglalapokat, ezek a részfolyamatok. Emelett van két piros karikánk. Ez a két piros karika amely jelzi a folyamatunknak a végét. Mivel mindkét halmaznak van egy adott teljes folyamata, ezért mindkét halmazt le kell zárni.

Természetesen ez a folyamat ábra nagyon egyszerű, a képen látható, hogy lehet belerakni gatewayeket, amelyek arra képesek hogy egy folyamatnak több elágazása legyen. Bár én nem illusztráltam a képen de lehet olyat is csinálni, hogy beszúrok egy gatewayt az étterem felhívásnál és ha ott adok egy olyan opciót, hogy az étterem zárva, akkor a teljes folyamatnak adhatok egy végpontot.

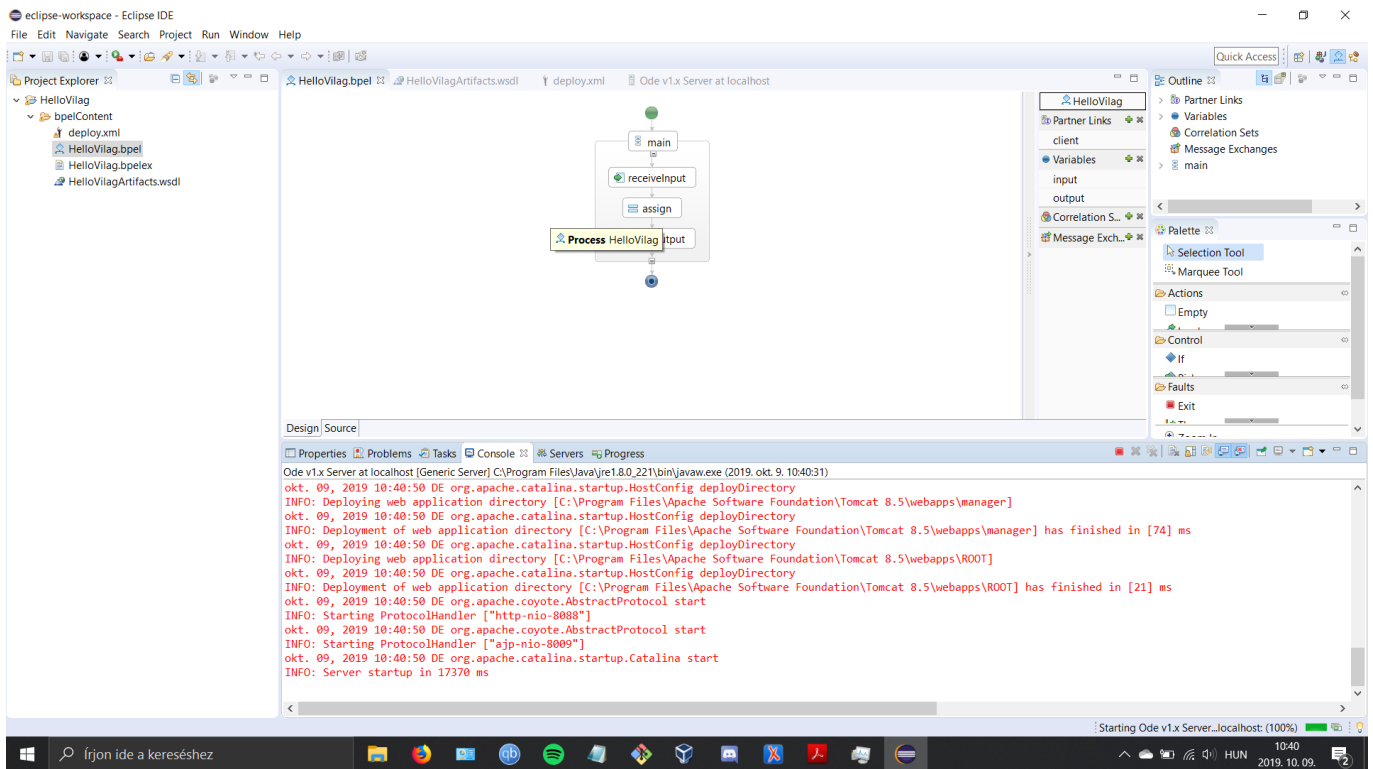
### 3.4. BPEL Helló, Világ! egy visszhang folyamat

BPEL (Business Process Execution Language) üzleti folyamatok modellezésének végrehajtó nyelve. XML alapú folyamatleíró nyílt szabványt alkalmaz. Elsősorban üzleti folyamatok leírására használatos, de egysegessége és elterjedtsége miatt sokszor használják munkafolyamatok leírását igénylő feladatokban is.

Minden egyes BPEL aktivitás egy külső, kiegészítő nyelven elkészített parancs meghívásával jár. A kiegészítő nyelv leggyakrabban Java, de lehet más, magas szintű script-nyelv is. A BPEL nyelvet a Microsoft és az IBM fejlesztette, illetve a korábbi BPML Üzleti folyamatokat modellező nyelvet használták fel.

Ebben a feladatban egy olyan webszervert kellett csinálni amelyben, ha egy string inputot adunk akkor a szervernek az output ugyan az a string legyen. A feladatot a Youtube-os link alapján csináltam, amely a pdf-ben található. Mivel ez a feladat "zöldes" (deprecated) eléggé nehéz volt a megtalálni a megfelelő szoftvereket találni. A legnagyobb problémám azzal adódott, hogy valamiért az Eclipse a szerverindításnál a portjaimat mindig foglaltak titálta. Végül sikerült a szervert elindítani és ezután már csak ki kellett próbálni, hogy működnek a funkciók.

Itt látható ezen a képen, hogy a szerver működik.



Valamilyen bug miatt az Eclipse-nél nem dobja fel a webservices opciót amivel tudnám a szimulálni a programomat. Ezt a hibát szeretném kijavítani később.

## 4. fejezet

# Helló Chomsky!

### 4.1. Encoding

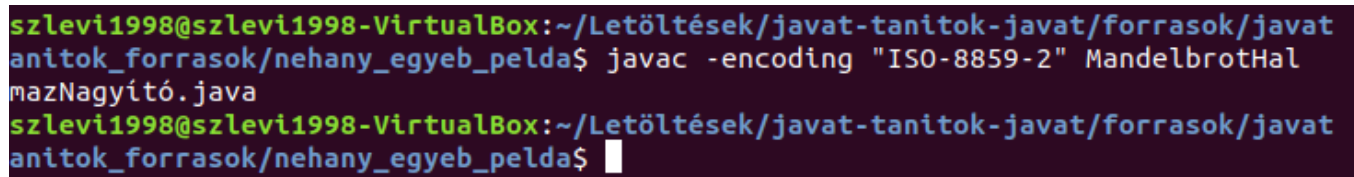
Ezt a feladatot az órán elkezdjük, azzal a problémával találkozunk, hogy a mandelbrotnagyító nem fordul le, az ékezetes betűkkel. Először is a problémánk az, hogy ne essünk abba a hibába mint én. Ami a PDF linkjében linkelt feladat önmagában sose fordul le, hiszen ez egy teljes projekt, nem csak egy programfájl. Amint fordítjuk a programunk még több hibával fog fordulni, de ez nem baj hiszen a hibák most már egyértelműek, ugyanis itt a java nem képes a magyar ABC szavait kezelni.

```
^
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációt:
    ^
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációt:
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
100 errors
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat
anitok_forrasok/nehany_egyeb_pelda$
```

Amint láthatjuk a képen, hogy az ékezetes betűk nem mappelhető karakterek. Ezek után utána kellett nézni, hogy milyen karakterkódolást kell alkalmaznunk, ahhoz hogy ezek a karakterek alkalmazhatóak legyenek. Én gyors keresés után a Java doksiknál megtaláltam azt, hogy magyar karakterek alkalmazásához encoding

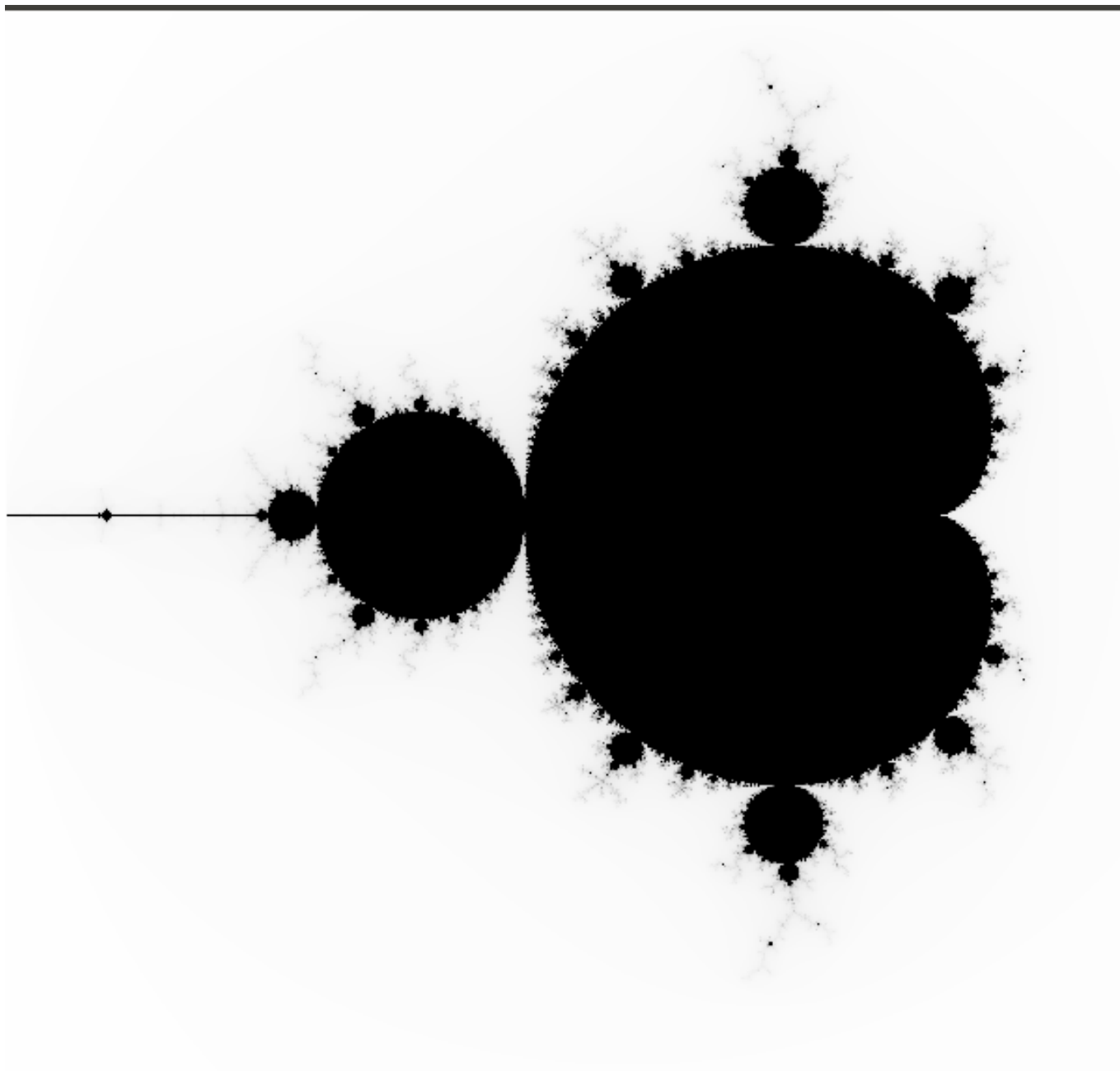


kapcsolót kell alkalmaznunk. A magyar karakterekhez a Latin 2-t kellett alkalmazni és ahhoz, hogy ezt működtessük, a fordításnál ezt a kapcsolót kell alkalmaznunk, az alábbi kép alapján.

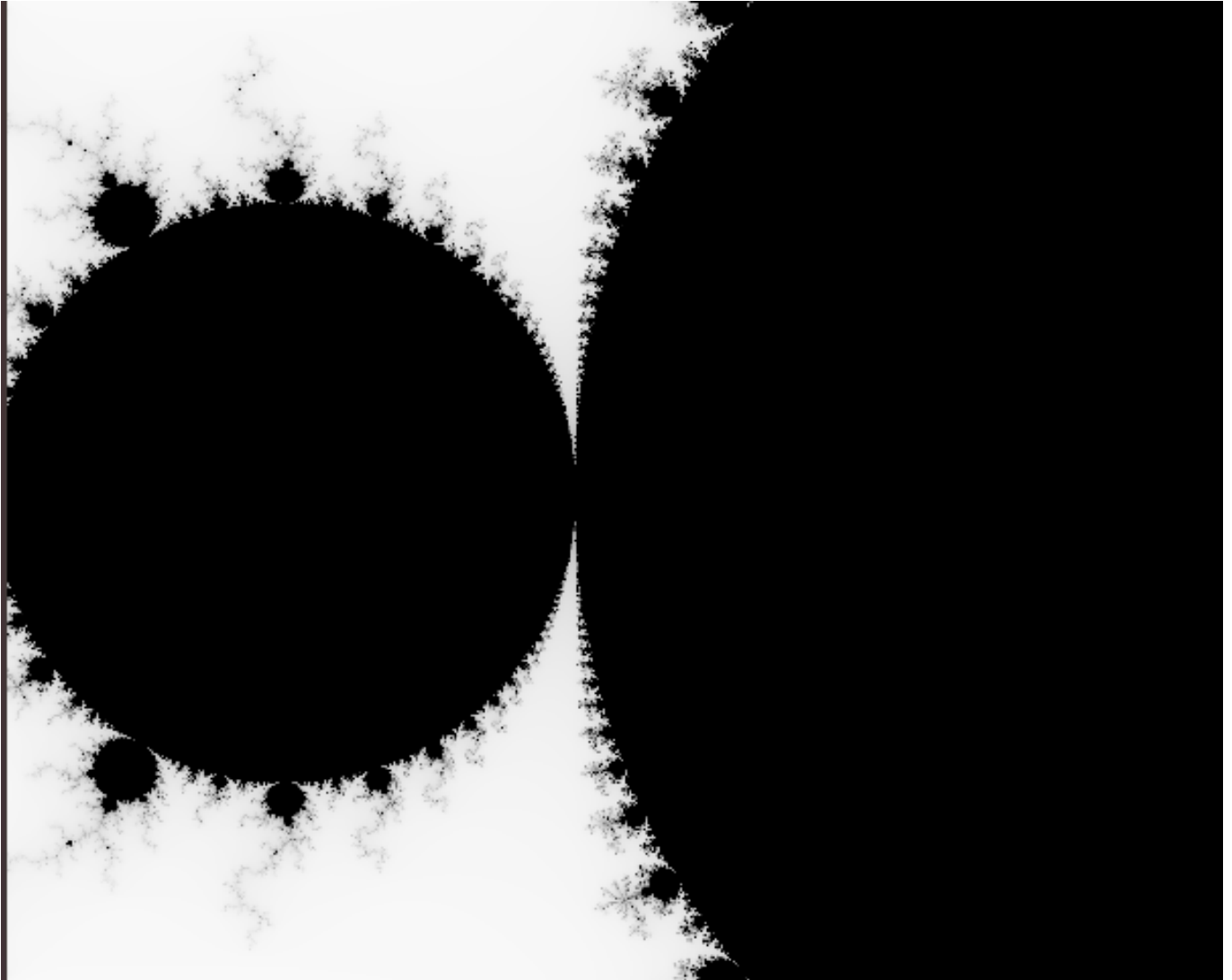
A screenshot of a terminal window with a dark purple background. The prompt is 'szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat'. The command entered is 'javac -encoding "ISO-8859-2" MandelbrothHal mazNagyító.java'. The prompt is repeated on the next line.

```
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat
anitok_forrasok/nehany_egyeb_pelda$ javac -encoding "ISO-8859-2" MandelbrothHal
mazNagyító.java
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat
anitok_forrasok/nehany_egyeb_pelda$
```

Itt a kép a fordításról.



Itt a kép a programról.



Itt a kép a futásról és a nagyításról.

Amint láthatjuk, a program ezek után gond nélkül fut és fordul. A képen a kedvünkre tudunk nagyítani, ott ahol akarunk.

## 4.2. Full screen

Ebben a feladatban az volt a lényeg, hogy egy olyan Java kódot írjunk amelyben, teljes képernyőt alkalmazunk.

```
import javax.swing.*;  
  
import java.awt.*;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;
```

```
public class Fullscreen {

    public static void main(String[] args) {

        JButton clickme = new JButton("Click me!");
        clickme.setBounds(180, 100, 200, 50);
        clickme.setFont(new Font("Times New Roman", Font.PLAIN, 11));

        JButton click = new JButton("Don't click me!");
        click.setFont(new Font("Times New Roman", Font.BOLD, 15));
        click.setBounds(400, 100, 250, 50);

        click.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                System.exit(0);
            }
        });

        JPanel panel = new JPanel();
        panel.setBackground(new Color(0, 191, 255));
        panel.setBounds(200, 200, 900, 300);

        panel.add(click);
        panel.add(clickme);
        panel.setLayout(null);

        JFrame frame = new JFrame("Fullscreen");

        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setUndecorated(false);
        frame.setVisible(true);
        frame.getContentPane().setLayout(null);
        frame.getContentPane().setBackground(new Color(255, 255, 0));
        frame.getContentPane().add(panel);

    }

}
```

Lássuk is a kódunkat.

```
        JButton clickme = new JButton("Click me!");
        clickme.setBounds(180, 100, 200, 50);
```

```
clickme.setFont(new Font("Times New Roman", Font.PLAIN, 11));
```

Ebben a részben létrehozok egy Java Butont amely clickmenek nevezek el. "" részben adtam meg, hogy a gombra mi legyen ráírva. Aztán a setBoundssal megadom a gombnak a helyzetét (x,y tengelyen) illetve a magasságát és a szélességét. Emellett változtatok a gomb betű típusán a méretét és a stílusát itt például dőltre.

```
JButton click = new JButton("Don't click me!");
click.setFont(new Font("Times New Roman", Font.BOLD, 15));
click.setBounds(400, 100, 250, 50);

click.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.exit(0);
    }
});
```

Itt létrehozok még egy gombot ez nagyon hasonlít a másik gombra,annyi különbséggel, hogy mássabb a betűstílus mérete és természetesen amit kiírok. MouseListener az egy olyan listener amely arra figyel, hogy ha rákattintok akkor az alkalmazást bezárja.

```
JPanel panel = new JPanel();
panel.setBackground(new Color(0, 191, 255));
panel.setBounds(200, 200, 900, 300);

panel.add(click);
panel.add(clickme);
panel.setLayout(null);
```

Itt létrehozok egy panelt amelynek a setboundssal adok méretet és pozíciót. Illetve a setBackgrounddal ennek a tálcának adok egy háttérszínt. A tálcához hozzáadom a gombokat a panel.add metódussal.

```
JFrame frame = new JFrame("Fullscreen");

frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setUndecorated(false);
frame.setVisible(true);
frame.getContentPane().setLayout(null);
frame.getContentPane().setBackground(new Color(255, 255, 0));
frame.getContentPane().add(panel);
```

```
}  
  
}
```

Ablakot itt hozzuk létre, a JFrame-mel és "Fullscreen" a cím viszont ezt nem láthatjuk, ugyanis a `frame.setUndecorated()` zároljuk, és ezzel nem látjuk a címsort. Azonban ezzel a metódussal, még nem változik teljes képernyőssé a programunk. Ezt `frame.setExtendedState(JFrame.MAXIMIZED_BOTH)`; ezzel a metódussal maximalizáljuk a szélességét és a magasságát az ablaknak.

Emelet még ahhoz, hogy ablak látható legyen a `frame.setVisible(true)`-t alkalmazzuk. Abban az esetben, ha `false` az értékünk, akkor nem láthatjuk az ablakunkat. Állítottam egy háttérszínt az ablaknak `frame.getContentPane().setBackground(new Color(255, 255, 0))`; -al és ezek után legutoljára, hozzáadtam a ablakhoz a tálcánkat, így válik teljessé a programunk.

### 4.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Ebben a feladatban egy OpenGL-es projektben kellett kisebb változtatásokat alkalmazni. Először is rengeteg dolgot felrakni, ahhoz hogy a programkódunk fusson és forduljon. Szükségünk volt a libboost-ra, az OpenGL-re és emelet számos update-ra. Ezek a szükséges szoftverek, könyvtárak után a fordítás és a futás sikeresen működött. Szokatlan volt az, hogy az irányítás ellentett volt. Először is rákerestem hogy, hol van a feladatban az irányítás része. Az adott programcsipetben látható a forgatásrésze.

```
void keyboard ( 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();  
}
```

Az eredeti kódban az értékek előtti műveleti jelek megvoltak cserélve, ezért ezeket átírtam az ellentetjére és ezek után kézre álló volt. Természetesen, ha az értékeken változtatunk akkor a forgatás mértéke is növekszik vagy csökken.

Emellett, az is feladatunk volt, hogy a változtassunk egy kicsit a színvilágon. Nem volt nehéz dolgunk hiszen a programunk nagyon sokszor alkalmazza az OpenGL-ben használt glColor3f függvényt. Ebben a függvényben 3 darab argumentum van és ezek az argumentumok adják meg a színünket. Az értékek az rgb-hez színvilágnak felel meg, hiszen a 3 érték a pirosnak a zöldnek és a kéknek felel meg. A számítása a következő : a választott értékeinket osztani kell 255-el és ezt a törtszámot kell nekünk megadni pl: aranyárgának az értékei :

```
glColor3f ( .960f, .772f, .0031f );
```

Ebbe a kis programcsipetben található a "sima" fehérén hagyott négyzeteket.

```
void drawPaRaCube ( int idx )
{
    glPushMatrix();

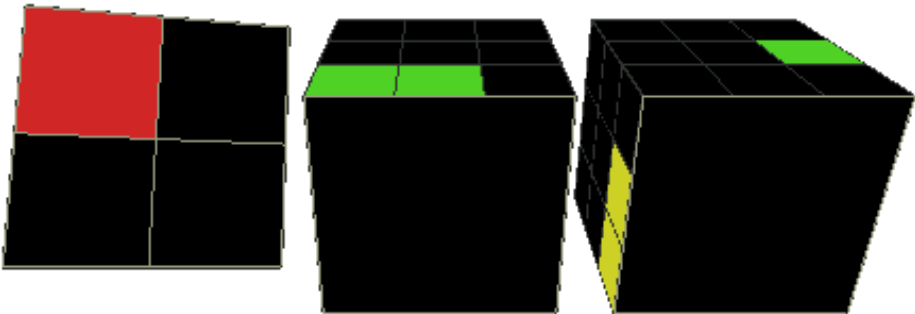
    int d = cubeLetters.size() /2 ;
    glTranslatef ( ( idx-d ) *2.5f, 0.0f, 0.0f );

    glRotatef ( cubeLetters[idx].rotx, 1.0f, 0.0f, 0.0f );
    glRotatef ( cubeLetters[idx].roty, 0.0f, 1.0f, 0.0f );
    glRotatef ( cubeLetters[idx].rotz, 0.0f, 0.0f, 1.0f );

    glBegin ( GL_QUADS );

    glColor3f ( .0001f, .0001f, .0001f );
```

Itt látható hogy a glColorba átállítottam, 0-ra és ezek után az eredetileg fehér négyzetek feketére változtak. A változások így szembetűnőek. Íme a kép:



DE



## 5. fejezet

# Helló, Stroustrup

### 5.1. JDK osztályok

Ebben a feladatban,azzal kellett foglalkoznunk, hogy egy olyan c++-os kódot kellett írunk,hogy a Java JDK osztályainak a fájlait kellett kiírtani. Ahhoz ,hogy ezt elérjük, először is szükségünk van egy Java JDK-ra amelyből ki tudjuk listázni a fájlokat. Ezután szükségünk van a libboostos könyvtárakra is.

```
sudo apt-get install libboost
```

Illetve a JDK osztályt lehetett volna tölteni az Oracle oldalára,de én nem akartam Oracle fiókot létrehozni, ezért innen töltöttem le : <https://bell-sw.com/pages/java-8u232/>

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

using namespace std;
using namespace boost::filesystem;

int main(int argc, char *argv[])
{
    path p ("src");

    if(!exists(p) || !is_directory(p)) {
        cout << p << "is not a path" << endl;

        return 1;
    }

    int fajlok = 0;

    recursive_directory_iterator begin(p), end;
```

```
vector <directory_entry> v(begin, end);
for (auto& f:v){
    if(path(f).has_extension()){
        cout << "File: " << path(f).filename() << endl;
        fajlok++;
    } else {
        cout << f << endl;
    }

}
cout << " A fájlok száma: " << fajlok << endl;
}
```

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

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

Lássuk is a kódot. Amint látható szükségem volt, a libboost osztályokra ezért includeolni kellett.

```
int main(int argc, char *argv[])
{

    path p ("src");

    if(!exists(p) || !is_directory(p)) {
        cout << p << " is wrong folder " << endl;

        return 1;

    }

    int fajlok = 0;
```

Ebben a részben a (mainben) először is megadjuk, hogy melyik mappát vizsgáljuk. Az ifben pedig azt vizsgáljuk, hogy megfelelő mappát vizsgáljuk. Abban az esetben ha nem jó mappát vizsgáljuk akkor, mint a kódban is látható kiadja a program, hogy nem jó mappát vizsgálunk.

A terminal window titled 'szlevi1998@Lenovo-Y520: ~/Letöltések'. The user runs the command 'g++ lib.cpp -o lib -std=c++11 -lboost\_system -lboost\_filesystem'. Then they run './lib', which results in the error message '"src" is wrong folder'.

```
szlevi1998@Lenovo-Y520: ~/Letöltések
szlevi1998@Lenovo-Y520:~/Letöltések$ g++ lib.cpp -o lib -std=c++11 -lboost_system -lboost_filesystem
szlevi1998@Lenovo-Y520:~/Letöltések$ ./lib
"src" is wrong folder
szlevi1998@Lenovo-Y520:~/Letöltések$
```

A képen is látható, hogy ha nem megfelelő path-t adok neki. Valamint adtam egy fájlok nevű változót amely arra szolgál, hogy majd később a fájlok számát ebbe a változóba tudjuk majd tárolni.

```
        recursive_directory_iterator begin(p), end;
vector <directory_entry> v(begin, end);
for (auto& f:v){
    if(path(f).has_extension()){
        cout << "File: " << path(f).filename() << endl;
        fajlok++;
    } else {
        cout << f << endl;
    }
}
cout << " A fájlok száma: " << fajlok << endl;
}
```

Ez a programnak az "agya". Ebben a részben használjuk azokat a libboost-os osztályokat amelyek miatt kellett includeolni a programunkba. A directory iterátorral végigjárjuk az összes directory entryt és amint. Ezeket a bejárásokat egy vectorba rögzítjük. Majd a for ciklusban az íffel megvizsgáljuk, hogy az adott fajlnak van-e utótagja, ha van akkor "File :" előtagot adok neki ezért megtudjuk különböztetni a fájlokat a mappáktól. Addig amíg a for ciklus fut, azaz addig amíg a bejárások megtörténnek addig növeljük a fajlok értékét. A fajlok növelése addig fut ameddig le nem fut a bejárás.

Ezután kiíratjuk, hogy hány darab fájlt találtunk.

```
szlevi1998@Lenovo-Y520: ~/Letöltések/5.óra
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$ g++ lib.cpp -o lib -std=c++11 -lboost
_system -lboost_filesystem
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$
```

```
szlevi1998@Lenovo-Y520: ~/Letöltések/5.óra
File: "launcher_ko.java"
File: "lib.cpp"
"src/launcher"
File: "jli_util.h"
File: "java.c"
File: "java_md.h"
File: "wildcard.h"
File: "version_comp.h"
File: "emessages.h"
File: "jli_util.c"
File: "manifest_info.h"
File: "defines.h"
File: "java_md_solinux.c"
File: "main.c"
File: "parse_manifest.c"
File: "java_md_common.c"
File: "java.h"
File: "splashscreen.h"
File: "splashscreen_stubs.c"
File: "version_comp.c"
File: "java_md_solinux.h"
File: "wildcard.c"
A fájlok száma: 17427
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$
```

## 5.2. Másoló-mozgató szemantika

Ebben a feladatban példákat kellett mutatni, olyan másoló-mozgató szemantikákra, amelyek a c++11-ben lettek elérhetőek. A Z3a9.cpp (ami tudatom szerint a legfrissebb binfa amit lehet találni).

Másoló és mozgató szemantika között elsődlegesen azon van a hangsúly, hogy jobbérték referencia kerül-e átadásra vagy balérték, mert jobbérték esetén a mozgató szemantika érvényesül, minden más esetben pedig a másoló konstruktor hívódik meg.

A másoló szemantika úgy inicializál egy objektumot, hogy egy másik objektumot használ fel a vele azonos osztályból. A dinamikusan lefoglalt változóknál nem elég a mutatókat másolni, új területet kell foglalni, és átmásolni a változó értékét. Az ilyen másolat készítését hívják mély másolásnak (deep copy), ellentétben azzal, amikor csak a mutatókat másoljuk.

```
LZWBinFa ( const LZWBinFa & regi ) {  
    std::cout << "LZWBinFa copy ctor" << std::endl;  
  
    gyoker.ujEgyesGyermeke ( masol ( regi.gyoker.egyesGyermeke () , regi ←  
        .fa ) );  
    gyoker.ujNullasGyermeke ( masol ( regi.gyoker.nullasGyermeke () , ←  
        regi.fas ) );  
  
    if ( regi.fas == & ( regi.gyoker ) )  
        fa = &gyoker;
```

Mint ahogy az outputban is látható, ez a binfa másoló konstruktor. Itt az történik, hogy az új gyökérnek az egyes és nullás gyermekeinek átadjuk a régi gyökeret.

```
LZWBinFa ( LZWBinFa && regi ) {  
    std::cout << "LZWBinFa move ctor" << std::endl;  
  
    gyoker.ujEgyesGyermeke ( regi.gyoker.egyesGyermeke() );  
    gyoker.ujNullasGyermeke ( regi.gyoker.nullasGyermeke() );  
  
    regi.gyoker.ujEgyesGyermeke ( nullptr );  
    regi.gyoker.ujNullasGyermeke ( nullptr );  
  
}
```

Ez pedig a mozgatókonstruktor a binfának. Itt az a lényeg, hogy a binfának a régi fáját mozgadjuk egy másik memóriacímre míg a régit töröljük. A régi gyökérnek az egyes és nullás gyerekeit is nullpointerrel tesszük egyenlővé, azaz lenullázzuk.

## 5.3. Változó argumentum ctor

Ebben a feladatban az volt a lényeg, hogy a prog 1-es példában is használt Perceptron osztály ne egy értéket adjon vissza, hanem egy ugyanakkora méretű képet kell visszatértenie.

Mielőtt a feladatra térnénk, fontos kiemelni, hogy a programhoz szükséges több fontos könyvtár ami lehetséges hogy nincs meg egyből. Ilyen például a libpng amely feltétlenül szükséges és fontos, bár nem tudom, hogy csak nálam okozott problémát, de szükséges a linkgrammar is.

Ennek a projektben 2 fontos fájlunk van és mind a két fájlban, kellett változtatásokat eszközölni. Először is foglalkozunk a main fájlunkkal.

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

using namespace std;

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, 1);

    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* kep = (*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].red = kep[i*png_image.get_width()+j];

    png_image.write("output.png");

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

Ez a main fájlunk. Ebben kellett egy kisebb változást alkalmazni.

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

Ebben a részben hozzáadjuk a fájlhoz a szükséges könyvtárakat. A png++ könyvtár teszi lehetővé, hogy képállománnyal tudjuk dolgozni.

```
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, 1);

    double* image = new double[size];
```

Itt van a main függvényünk. Az első sorban meghatározzuk, hogy a képállományunk az parancsori argumentum legyen. Ezután létrehozuk a size int típusú változót. Ez a változó lesz, amellyel kiszámoltatjuk a képnek a méretét és ebbe fogjuk tárolni az adott méretet. Ezután példánosítjuk a Perceptron osztályt. Majd létrehozunk egy double mutatót.

```
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;
```

Itt ezután létrehozunk két for ciklust amelynek, a funkciója az, hogy a 2 ciklus egyenként átmennek a kép magasságán, illetve a kép szélességén és ezt majd később az imagebe fogja tárolni.

```
double* kep = (*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].red = kep[i*png_image.get_width()+j];

png_image.write("output.png");

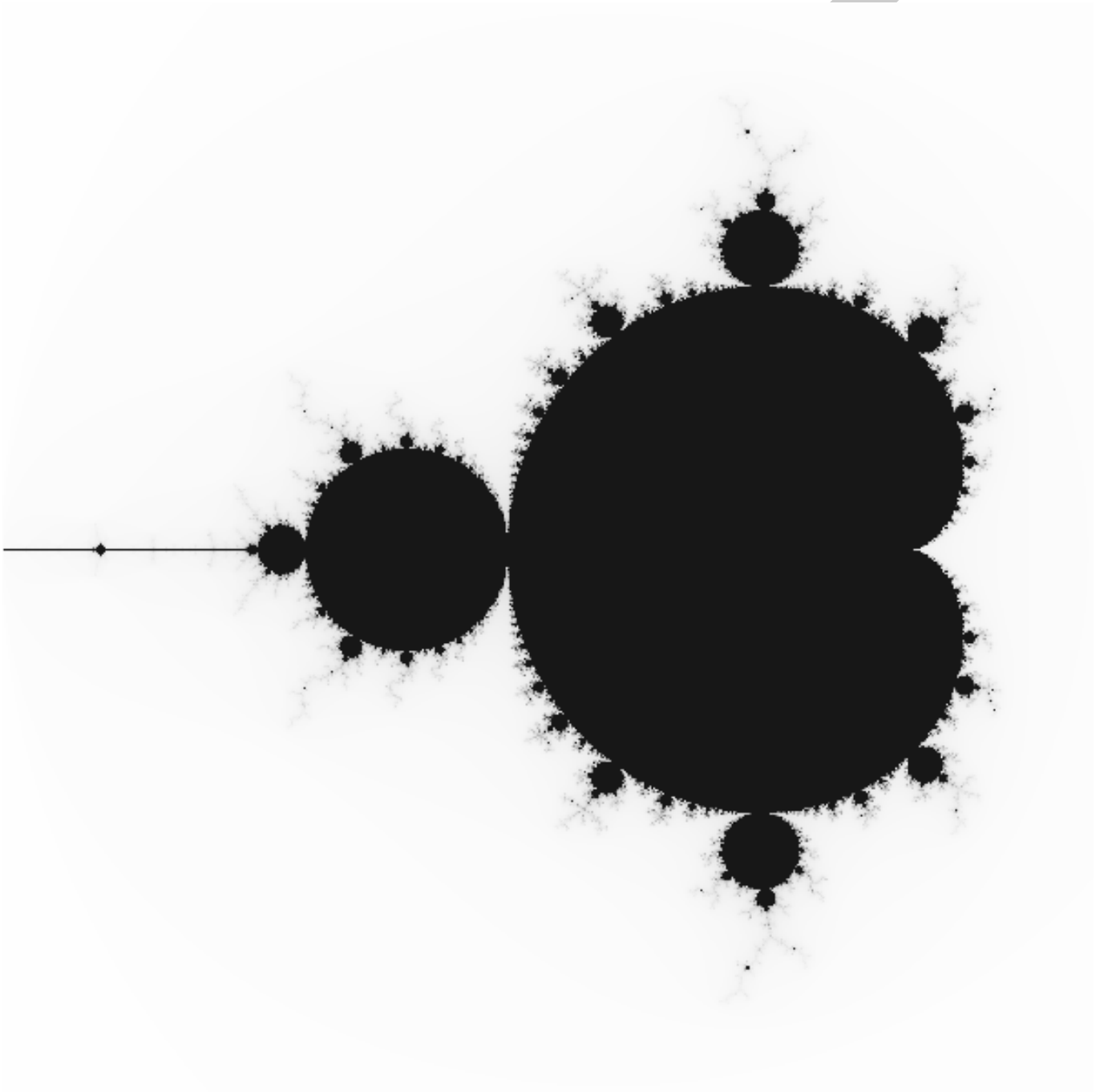
delete p;
delete [] image;
}
```

Itt változtatunk igazán a programon, hiszen az image-al már nem értéket fog visszaadni, hanem most már egy képet. Itt is double csillag típusra van szükség. Itt ismét 2 for ciklus kell amely átmegy az eredeti képnek szélességén és magasságán és ezeket az értékeket majd a kep be tároljuk. Ezután alkalmazzuk a write függvényt és ezzel a tárolt adatok alapján egy új output.png-t létrehozunk. Ezzel még nem vagyunk

készek, hiszen az mlp fájlban át kell írni az operator függvénynek a visszatérítési értékét, ugyan is már nem double-t kell visszatérítenie, hanem egy double mutatót kell.

```
// double operator() ( double image [] )  
double* operator() ( double image [] )
```

Így kellett változtani az mlp.hpp-n és ezek után már gond nélkül működik a programunk. Íme a kép a programunkról:



Amint látható a másolás tökéletesen működött, a mandel és az output teljesen megegyezik.



## 6. fejezet

# Helló, Gödel

### 6.1. Gengszterek

Ebben a feladatban a robocar lambda kifejezésével ismerkedünk meg. A lambda kifejezések olyan függvények, amelyeket inlineoknak nevezhetünk. Ennek az a jelentősége, hogy ezeket a függvényeket csak egyszer alkalmazzunk az adott kódcsipetben. A szintaxisa a következő:

```
[ bemeneti ág ] (paraméterek) -> visszatérítési érték
{
    metódus tartalma
}
```

Amit még érdemes tudni a lambda függvényekről, hogy ezekben a függvényekben a fordító is megadhatja a visszatérítési értékeket.

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

Ezt a részt kellett elemeznünk a feladatunkban. Amint láthatjuk, hogy a sort függvényünknek 3 darab paramétere lesz. Az első két paraméternek a gangsters.begin() és a gangster.end() az lesz a szerepe, hogy a teljes gangster vectort végigvizsgálja. A 3. paraméter képezi a lambda függvényt. A bemeneti ágon kiválasztjuk a 2 változót ami a this és a cop objektum. A lambda függvényünk paraméterként a Gangster x-et és a Gangster y-t várja. Ez a két paraméter lesz összehasonlítva.

```
    return dst ( cop, x.to ) < dst ( cop, y.to );
}
```

Itt azt adja vissza, hogy az x vagy az y van közelebb a cophoz. Abban az esetben, ha az x közelebb akkor true értéket ad vissza.

## 6.2. STL map érték szerinti rendezése

Ebben a feladatcsokorban a mappekkel is kellett foglalkoznunk és az volt a feladatunk, hogy c++-ban olyan programot írjunk amellyel a mapben ne a kulcs alapján rendezzünk, hanem érték alapján. A feladatban az idei F1-es csapatok pontszerzéseit rendeztem. Itt a kód:

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

int main()
{
    std::map<std::string, int> Championship = { { "Ferrari", 479} , { "Williams", 1}, { "Red Bull", 366}, { "AlfaRomeo", 35}, { "McLaren", 121}, { "RacingPoint", 65}, { "Mercedes", 695} , { "ToroRosso", 64}, { "Haas", 28}, { "Renault", 83} };

    typedef std::function<bool(std::pair<std::string, int>, std::pair<std::string, int>)> Compare;

    Compare lambdafugg =
    [](std::pair<std::string, int> point1, std::pair<std::string, int> point2)
    {
        return point1.second > point2.second;
    };

    std::set<std::pair<std::string, int>, Compare> sorted(
        Championship.begin(), Championship.end(), lambdafugg);

    for (std::pair<std::string, int> element : sorted){

        std::cout << "Team name : " << element.first << "\t" << " Points scored : " << element.second << std::endl;
    }
    return 0;
}
```

Elemezzük ki a kódot

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

int main()
{
```

```
std::map<std::string, int> Championship = { { "Ferrari", 479} , { "←  
Williams", 1}, { "Red Bull", 366}, {"AlfaRomeo", 35}, {"Mclaren", 121}, ←  
{ "RacingPoint", 65}, { "Mercedes", 695} , {"ToroRosso", 64}, {"Haas", ←  
28}, {"Renault", 83} };
```

```
typedef std::function<bool(std::pair<std::string, int>, std::pair<std:: ←  
string, int>)>Compare;
```

Amint láthatjuk, nem kevés header fileokat kell alkalmazni. A mapre azért van szükségünk, mert először is ez volt a feladat alappillére, illetve azért is mert Mappel tudunk érték-pár változókat létrehozni. A setre és a functionalre később van szükségem. A mainben létrehozuk a Mapet amelyben egy string és egy int páros van. Mapet elnevezzük Championshipre és ezután feltöltöm a mapet a csapatok nevével és a hozzá tartozó pontokat. Ezután egy típust definiálunk amelynek az a lényege, hogy 2 párt fogad be és ez visszaad egy bool értéket amelyet a Compare tárol.

```
Compare lambdafugg =  
[] (std::pair<std::string, int> point1, std::pair<std::string, int> point2 ←  
)  
{  
    return point1.second > point2.second;  
};
```

Ezután létrehozok egy lambdafüggvényt, és ebben a függvényben vizsgálunk 2 elemet. Ezután visszaadunk egy igaz vagy hamis értéket.

```
std::set<std::pair<std::string, int>, Compare> sorted(  
Championship.begin(), Championship.end(), lambdafugg);
```

Itt a seten belül történik a rendeződés. A setben meghatározzuk azt hogy melyik osztályt rendezzük.

```
for (std::pair<std::string, int> element : sorted){  
  
    std::cout << "Team name : " << element.first << "\t" << " Points scored ←  
: " << element.second << std::endl;  
}  
return 0;
```

Ezután a for ciklussal kiírom a csapatok nevét és a csapatok pontszámát.

```
szlevi1998@Lenovo-Y520: ~/Letöltések/6.óra/map
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ g++ -std=c++11 mapping.cpp -o map
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ ./map
Team name : Mercedes      Points scored : 695
Team name : Ferrari       Points scored : 479
Team name : Red Bull      Points scored : 366
Team name : McLaren       Points scored : 121
Team name : Renault       Points scored : 83
Team name : RacingPoint   Points scored : 65
Team name : ToroRosso     Points scored : 64
Team name : AlfaRomeo     Points scored : 35
Team name : Haas          Points scored : 28
Team name : Williams      Points scored : 1
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$
```

Abban az esetben ha csökkenőre szeretnénk állítani az értékeket akkor egy apró módosításon kell túllépni.

```
return point1.second < point2.second;
```

Amint változtatunk a relációjelen a sorrendünk megváltozik.

```
szlevi1998@Lenovo-Y520: ~/Letöltések/6.óra/map
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ g++ -std=c++11 mapping.cpp -o map
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ ./map
Team name : Mercedes      Points scored : 695
Team name : Ferrari       Points scored : 479
Team name : Red Bull      Points scored : 366
Team name : McLaren       Points scored : 121
Team name : Renault       Points scored : 83
Team name : RacingPoint   Points scored : 65
Team name : ToroRosso     Points scored : 64
Team name : AlfaRomeo     Points scored : 35
Team name : Haas          Points scored : 28
Team name : Williams      Points scored : 1
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ gedit map
map
mapping.cpp
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ gedit mapping.cpp &
[2] 21594
[1] Kész          gedit mapping.cpp
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ g++ -std=c++11 mapping.cpp -o map
[2]+ Kész        gedit mapping.cpp
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$ ./map
Team name : Williams      Points scored : 1
Team name : Haas          Points scored : 28
Team name : AlfaRomeo     Points scored : 35
Team name : ToroRosso     Points scored : 64
Team name : RacingPoint   Points scored : 65
Team name : Renault       Points scored : 83
Team name : McLaren       Points scored : 121
Team name : Red Bull      Points scored : 366
Team name : Ferrari       Points scored : 479
Team name : Mercedes      Points scored : 695
szlevi1998@Lenovo-Y520:~/Letöltések/6.óra/map$
```

### 6.3. Alternatív Tabella

Ebben a feladatban az alternatív tabellával foglalkozunk. Az alternatív egy olyan tabella, amely a csapatok a valós erőssorrendjét mutatja meg. Ennek az a lényege, hogy megnézi milyen csapatok ellen érte el az adott eredményt. Ez az értékelés arra jó, hogy megmutassa hogy 3 győzelem a 3 leggyengébb csapata ellen, nem olyan nagy teljesítmény mint, ha a 3 győzelem a 3 legerősebb csapat ellen. Egy sima tabellában mindkét esetben 9 pontot ér, de ezzel az algoritmussal megtudjuk különböztetni a csapatok valós teljesítményét.

```
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 a kódcsipetet kellett elemeznünk és azon belül a Comparable interfésszel. Először is létrehozunk egy csapat osztályt. Ezután a csapat osztály implementálja a Comparable interfészt. Ezután megvizsgáljuk, hogy vajon összehasonlítható két csapat. Objektumot hasonlítunk össze az átvett objektummal. 3 lehetőségünk van, vagy -1, 1 vagy 0 lehetséges. Ez azt jelenti, hogyha -1 az érték akkor az objektum nagyobb mint az átvett az érték. Ha 1 az érték akkor az objektum kisebb mint az átvett érték és végül, ha 0 akkor a két objektum értéke megegyezik.

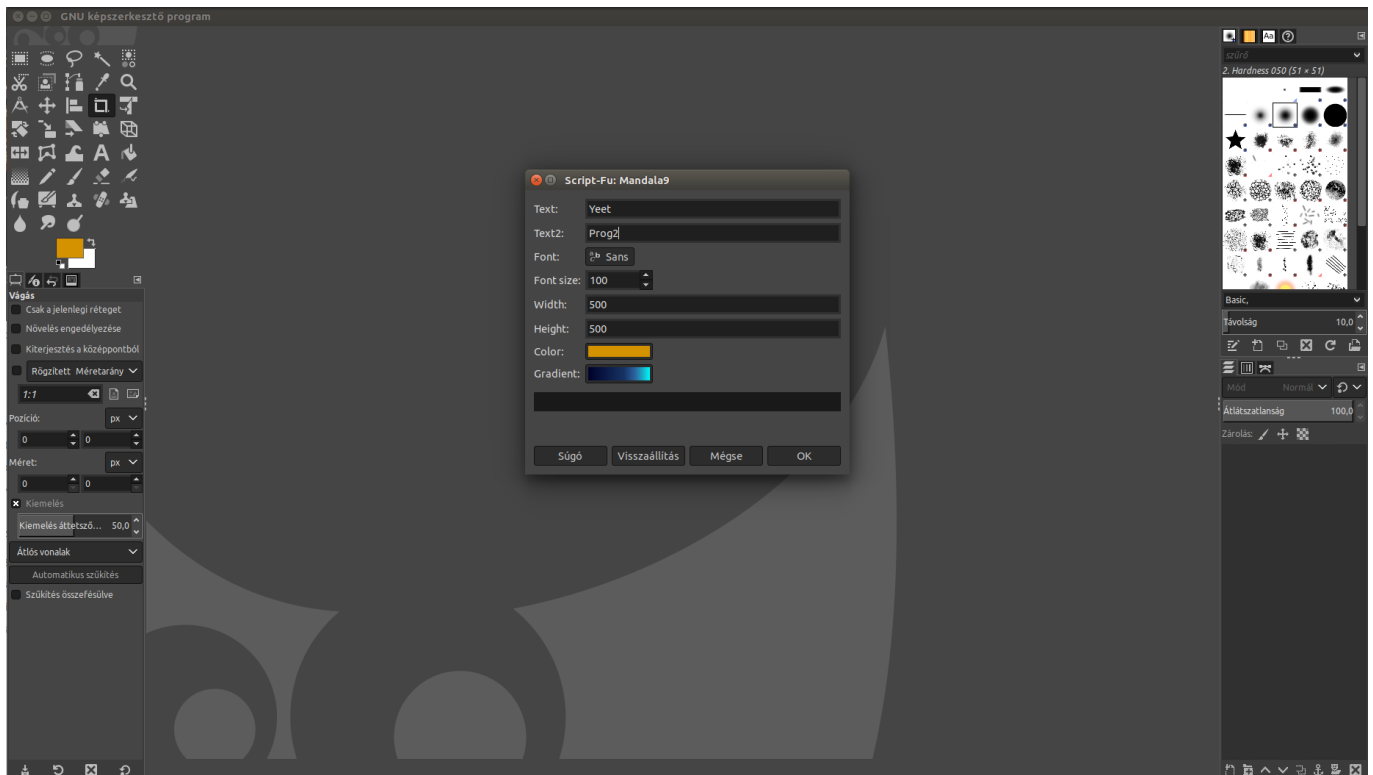
## 6.4. GIMP Scheme hack

Ebben a feladatban, Bátfai tanárúr GIMP-es példáját kellett alkalmaznunk. Ez a feladat prog1-ben, volt viszont eddig még nem csináltam, ezért most fogom megcsinálni.

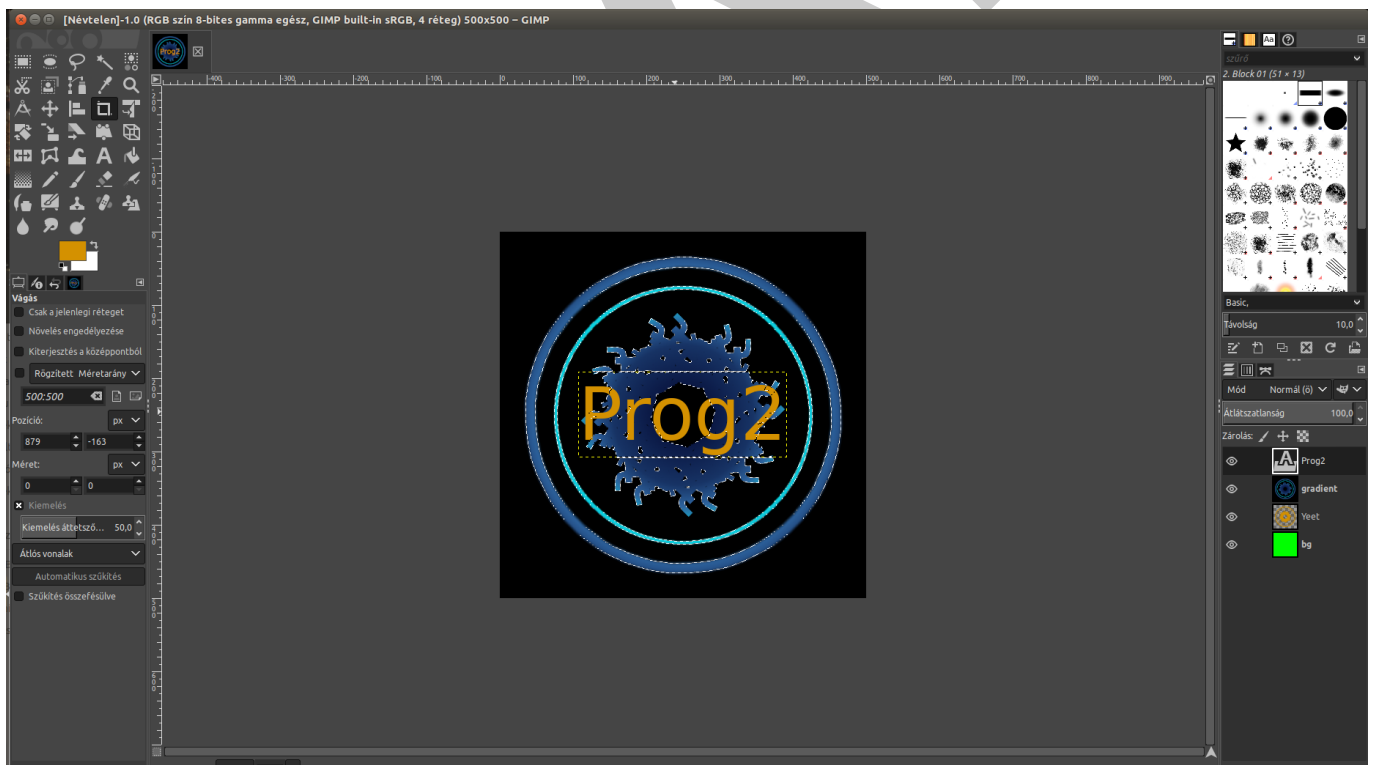
A feladat elkészítéséhez kellett alkalmazni a GIMP-et. Tanár úrnak a scriptjét alkalmazzuk. A legfontosabb dolog, hogy a scriptet a megfelelő helyére helyezzük. Egészen pontosan ide:

```
/home/someusername/snap/gimp/227/.config/GIMP/2.10/scripts
```

Amint ha jól elhelyezzük a scriptet, azután a GIMP-ben alkalmazhatjuk. Ahhoz, hogy láthassa a GIMP a projektet a szűrőkben a rendszerfájlokat frissíteni kell. Ezután újra elindítjuk és ezután már létrehozhatjuk a saját mandalánkat. Itt lehet a kedvünkre állítani, hogy mekkora legyen a kép mérete, milyen típusú legyen a szövegtípus, milyen színű legyen a szöveg.



És erre a beállításokra, ilyen mandalát kapunk vissza.



```
(script-fu-register "script-fu-bhax-mandala"  
"Mandala9"  
"Creates a mandala from a text box."  
"Norbert Bátfai"  
"Copyright 2019, Norbert Bátfai"  
"January 9, 2019"
```

```
" "  
SF-STRING      "Text"      "Bátf41 Haxor"  
SF-STRING      "Text2"     "BHAX"  
SF-FONT         "Font"      "Sans"  
SF-ADJUSTMENT  "Font size" '(100 1 1000 1 10 0 1)  
SF-VALUE       "Width"     "500"  
SF-VALUE       "Height"    "500"  
SF-COLOR       "Color"     '(212 145 0)  
SF-GRADIENT    "Gradient"  "Deep Sea"
```

A fájlban itt található meg az alapbeállítások. Abban az esetben ha nem akarunk már változtani a projekten, akkor érdemes a forrásban átírni ezeket az adatokat, hiszen ilyenkor már csak a mandala létrehozása lenne hátra.

## 7. fejezet

# Helló, Valaki!

### 7.1. FUTURE tevékenység editor

Mielőtt a feladathoz térnénk, fontos kiemelni, hogy a program megfelelően működhessen, szükségünk van egy jó JDK verzióhoz és egy JavaFX verzióhoz. Enélkül az eredeti program 100 hibával fog kilépni. Ahhoz, hogy feltelepítsük a JavaFX-t ezt a parancsot kell alkalmaznunk.

```
sudo apt install openjdk-8-jdk openjfx
```

Valamint én még a JDK-t is defaultra változtattam, igazából nem tudom, hogy szükséges volt ahhoz, hogy működjön, mert nem tudtam, hogy milyen JDK volt alapból a rendszeremen, de inkább biztosra mentem. A JDK frissítéséhez az alábbira parancsra van szükségünk.

```
sudo apt-get install default-jdk
```

Feladatunk az volt, hogy a Future Activity Editorba javítsunk valami féle hibát, illetve keressünk benne bugot. Nem kellett sok idő, ahhoz hogy bugot találjunk. Legegyszerűbben megtalálható bug az volt, hogy egy adott fa elemhez nem lehet egyszerre több altevékenységet hozzárendelni. Itt láthatjuk a képen is.



```
szlevi1998@Lenovo-Y520: ~/Letöltések/Prog2/7.óra/future-master/cs/F6
szlevi1998@Lenovo-Y520:~/Letöltések/Prog2/7.óra/future-master/cs/F6$ javac ActivityEditor.java
szlevi1998@Lenovo-Y520:~/Letöltések/Prog2/7.óra/future-master/cs/F6$ java ActivityEditor
Cannot create City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új CoC/Új altevékenység
```

Nézzük is a kódot.

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

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);
} else {

    System.err.println("Cannot create " + f.getPath());

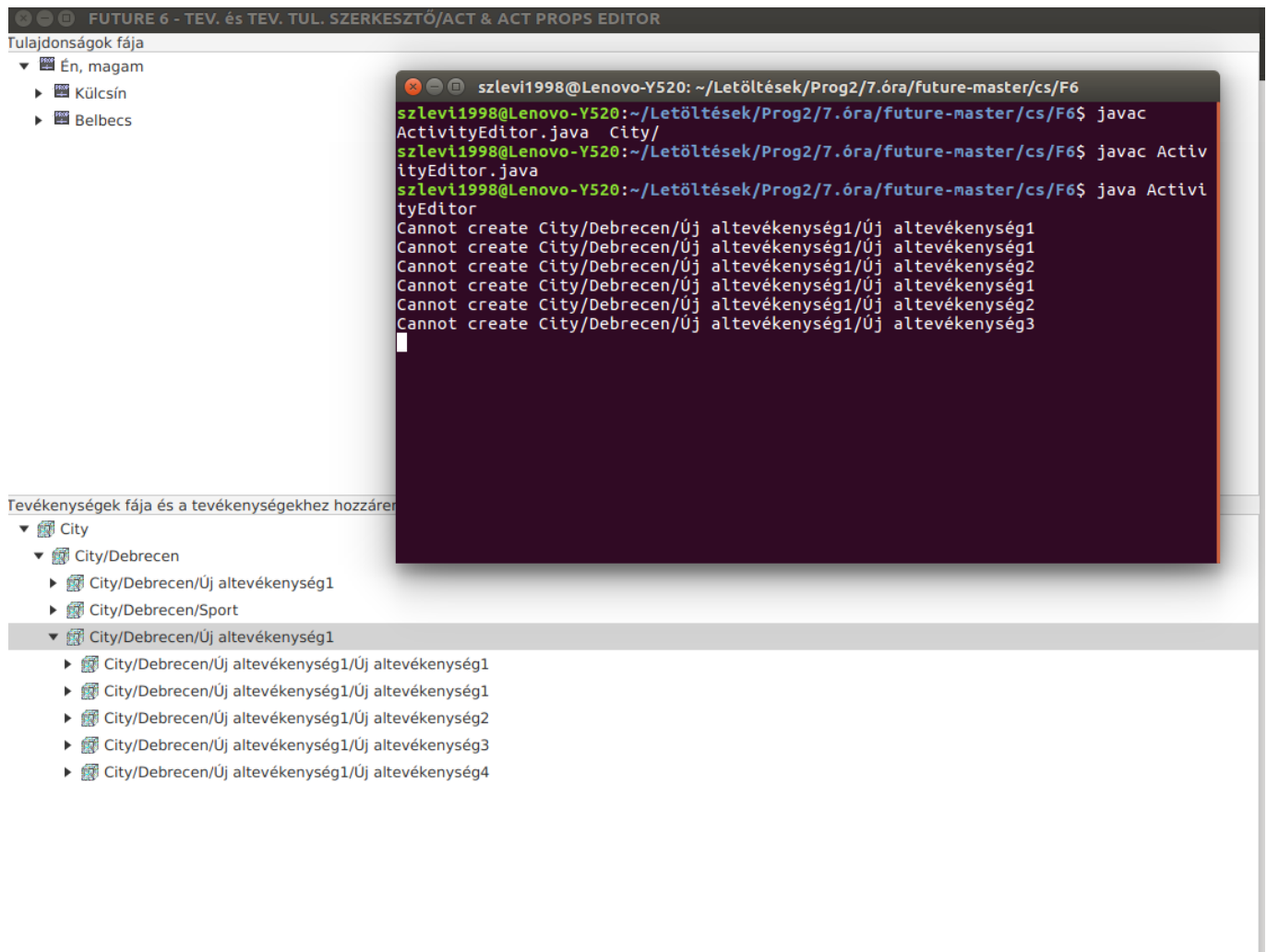
}
});
```

Ebben a kódcipetben tudunk létrehozni altevékenységeket. Ezzel az a probléma, mint említettem, hogy egy adott altevékenység csak 1 altevékenységet lehet létrehozni, többet nem lehetséges. Viszont, ha kicsit módosítunk a kódon, akkor ki lehet küszöbölni ezt. Itt az alábbi módosítások.

```
int i = 1;
while(true) {
    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());
        }
    }
});
```

Itt létrehozok egy `i` változót és ezután létrehozok egy while ciklust. A while ciklusban a paraméterben egy `true`-t adok meg ezzel végtelen ciklust hozok létre. Ebbe a végtelen ciklusba helyezzük az altevékenység létrehozó részét. Az `if`-ben megnézzük hogy létezik már ilyen altevékenység abban az esetben ha nincs akkor létrehozunk egy újat, ha van akkor az `if` függvényünket `break`eljük (megszakítjuk). Fontos, hogy a `break` benne legyen ebben a ciklusba, hiszen ha nem teszünk, akkor egészen biztosan megfagyasztjuk a programunkat és eléggé esélyes az is, hogy a gépünk is lefagyhat. Ha van ilyen mappánk akkor növeljük az `i` változónkat és kiíratjuk, hogy nem sikerült ilyen fájlt létrehoznunk. A programról egy kép:



## 7.2. SamuCam

Ebben a feladatban az volt a feladatunk, hogy SamuCam-et felélesszük és rámutassunk a webcam kezelésére. A felélesztés egy kicsit idegesítő procedúra volt, hiszen abban a tudatban voltam, hogy minden könyvtáram valid és rendben van azonban csak később jöttem rá, hogy a Samucam header fájlában includeolt könyvtárak már nem frissek. Eredetileg így nézett ki.

```
#include "opencv2/objdetect.hpp"  
#include "opencv2/core.hpp"  
#include "opencv2/highgui.hpp"  
#include "opencv2/imgproc.hpp"
```

```
#include "opencv2/objdetect/objdetect.hpp"  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"
```

Így néz ki helyesen a könyvtár szerkezete, így működik megfelelően.

```
#include "SamuCam.h"

SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = 144 )
: videoStream ( videoStream ), width ( width ), height ( height )
```

Includeoljuk a SamuCam.h headerbe. Itt megadjuk az IP cím által használt kamerának a videóstreamnek a szélességet és magasságát.

```
{
    openVideoStream();
}

SamuCam::~SamuCam ()
{
}

void SamuCam::openVideoStream()
{
    videoCapture.open (0);

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 30 );
```

VideoCapture.open metódusba megadjuk az argumentumba egy 0-ás számot amely egy ID számot takar és ez az ID szám videocapture a webcam-re mutatott. Ezután a settel beállítjuk a szélességét a magasságát és a FPS-nek a számát.

```
{

    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;
    }
}
```

Ezután a CascadeClassifert objektumot példányosítjuk. Ezután beolvassuk az lbpcascade\_frontalface.xml-t. Classifier arra alkalmasa hogy az arc képünket tudja elemezni. A load metódussal olvassuk be az arcot. Akkor,ha nem tudja beolvasni akkor hibát ad vissza a program.

```
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, 3, ↵
            0, 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 );
    }
}
```

```
    }

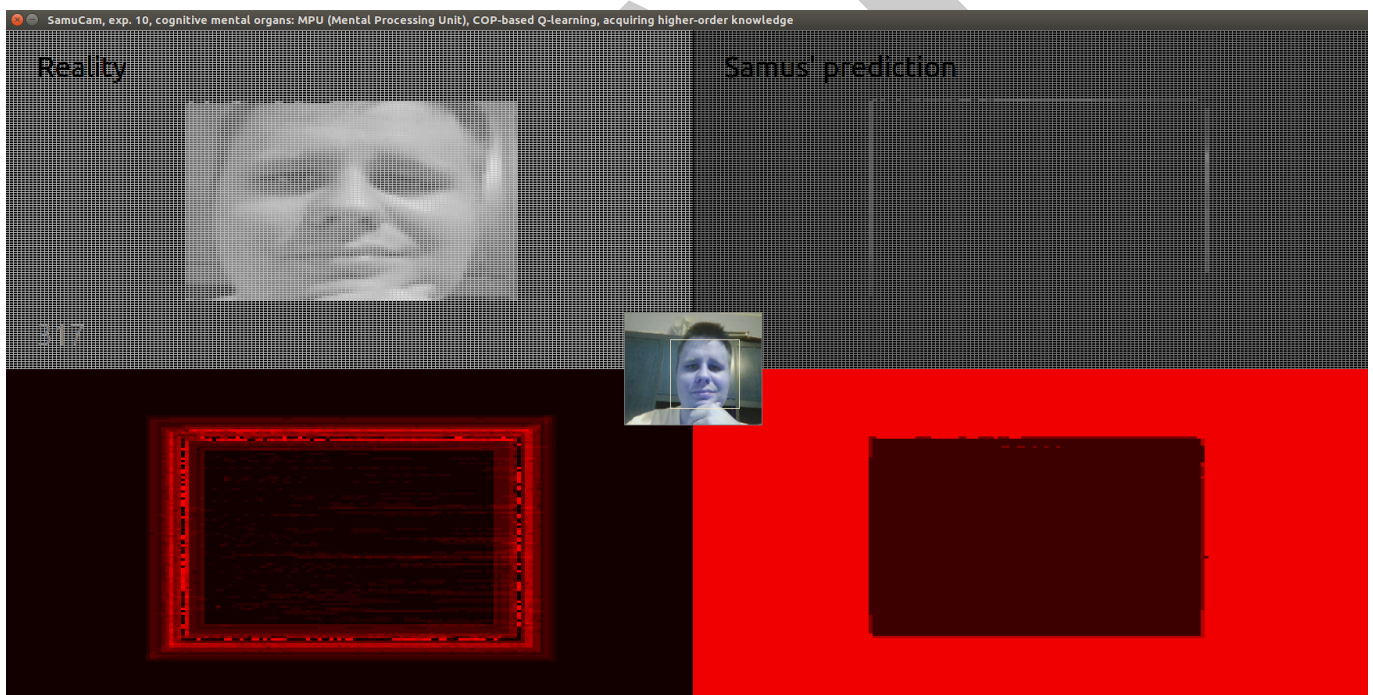
    QThread::msleep ( 80 );

}

if ( ! videoCapture.isOpened() )
{
    openVideoStream();
}

}
```

Ebben a kódcsipetben 2 while ciklust alkalmazunk. Létrehozunk egy frame változót amely egy tömb. Egy if függvény paraméterében vizsgáljuk, hogy a frame üres vagy nem. Ha a frame nem üres akkor átméretezzük a képet. CvtColoral átszínezzük a képünket szürkésre és equalizeHisttel pedig kiegyenlítjük a hisztogramot. Majd ezután a detectMultiScalel a képeket elmentjük a facesbe. Ha találunk képet akkor létrehoz egy QImage-t és ezt a SamuBrainnek továbbítja amely később feldolgozza. Ezután 80 milliszekundum után visszatérünk a while ciklus elejére.



Először IP alapján próbáltam működtetni, majd később a videocapture.open(0)-ra állítottam és amint látható a program működik.

## 7.3. BrainB

Ebben a feladatban az volt a feladatunk, hogy a Brain B ismertessük a QT-nek a Slot- Signal mechanizmusát. Mielőtt a feladathoz térnénk, fontos ismertetni azt, hogy mi is a slot signal mechanizmus, mire használjuk. Ahhoz, hogy megértem, ezt a mechanizmust a QT-nek a dokumentációját kellett értelmezni. A QT-nek a dokumentációját itt találtam : <https://doc.qt.io/qt-5/signalsandslots.html>

Először is Signalokat és a slotokat arra használjuk, hogy az objektumokat között kommunikálhassunk. A Qt-nek ez a legfőbb mechanizmusa. A QT-ben egy signal akkor van aktiválva amikor egy bizonyos esemény történik. Qt-nek vannak előre definiált signaljai, viszont alkalmazhatunk alosztály moduljait amit hozzáadhatunk a saját signaljainkhoz. A slot egy függvény amely válaszként hívódik meg egy bizonyos signal alapján.

A QT signalok és slotoknak a mechanizmusa biztosítja azt, ha egy signalt kapcsolunk egy slothoz akkor a megfelelő időben a slot meghívódik a signal paramétereivel. A signalok és a slotoknak bármilyen számokat és bármilyen típust képes elfogadni az argumentumában. Ezért ezek a típusok biztonságosak.

Minden osztály ami örököl az QObjectból vagy egy bizonyos subclassnak pl QWidgete tartalmazza signalokat és a slotokat. Signalok akkor hívódnak meg objektumoktól, ha valamikor megváltozik az állapotuk egy olyan mértékben amely már a többi objektumot is befolyásolhatja. Az objektumok csak kommunikálnak. Nem tudja vagy nem érdekli az, ha valamilyen signalt fogad akkor azt tovább sugározza. Ez teljes információs egyedbefoglalás és biztosítja, hogy egy objektumot használhatunk akár egy szoftveres komponensnek.

Slotokat mint említettük használhatjuk hogy signalokat fogadjunk, de ezek lehetnek csak sima függvények. Egy slothoz több signalt kapcsolhatunk és egy signalt több slothoz is kapcsolhatunk. Az is lehetséges hogy egy signalt egy másik signalhoz kapcsolunk. A signalok és a slotok együttesen egy nagyon erős programozói mechanizmust alkot. a feladról.

Brain B-ben alkalmazzuk a BrainWin.cpp-ben. Signalokat a slotokkal egy connect függvénnyel kapcsoljuk össze. Első signal függvényénél megadjuk hogy a heroesChangedet. A függvényben azt adjuk meg hogy milyen paramétereket várunk a heroeschangedtől majd ezt továbbítjuk a slothoz és majd megadjuk hogy fusson le az updatehereos függvény. A második is hasonlóan működik csak ott kevesebb paramétert használunk. Az első függvény akkor fut le, ha a Brain B-ben az indulástól számítva 10 perc letelik. A második függvény pedig folyamatosan fut és az arra van, hogy az entropik frissüljenek.

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

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

## 8. fejezet

# Helló, Gödel

### 8.1. Port Scan

Ebben a feladatban egy olyan programot néztünk, amelyben portokat vizsgáltunk. Itt Java socketekkel dolgozunk. A socket egy végpont a kommunikációban két gép között. A socketnek a dolgát SocketImpl osztálynak példánya végzi. Egy alkalmazás amely változatunk az létrehoz egy socket implementációt. Ez a socket implementáció magától létrehozhat további socketeket amelyek megfelelőek a helyi tűzfalnak. Amint látható for ciklusban végezzük a vizsgálatot.

```
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 + " figyel!");  
  
                socket.close();  
  
            } catch (Exception e) {  
  
                System.out.println(i + " nem figyel!");  
  
            }  
  
        }  
  
    }  
}
```

Minden egyes portot vizsgáljuk és, ha az adott portot vizsgálja akkor kiírja a program, hogy ezt a portot figyel. Akkor, ha nem figyel akkor a függvényt kivételt kezel és ki írja hogy nem figyel. Ahhoz, hogy ez működjön fontos, hogy a terminálba a paraméternek a localhostot kell adni. A következőképpen:



```
java KapuSzkenner localhost
```

Itt adjuk meg a futtatásnak a módját. Amint lehet látni a localhostot adjuk meg és 1 portot figyel míg az összes többi nem figyelte.

```
623 nem figyel
624 nem figyel
625 nem figyel
626 nem figyel
627 nem figyel
628 nem figyel
629 nem figyel
630 nem figyel
631 figyel
632 nem figyel
633 nem figyel
634 nem figyel
635 nem figyel
636 nem figyel
```

## 8.2. Android Játék

Ebben a feladatban Androidos játékot kellett készítenünk. Tavalyi Prog 2-ben ugyanezt feladatot megoldtam úgy, hogy csak átmentettem a tavalyi példámát. Ezt a feladatot Learn the Java Easy Way könyvből vettem át illetve módosítottam egy kicsit a kódon is.

A feladat lényege azt volt, hogy egy adott számot kellett kitalálni 1 és 100 között. Akkor, ha szám alacsonyabb akkor a program kiírja, hogy adjunk egy nagyobb számot, illetve ez természetesen fordítva is működik. Ha eltaláljuk a számot akkor a program lefut és ad egy "gratulációs üzenetet."

```
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Button;
import android.widget.TextView;
import org.w3c.dom.Text;

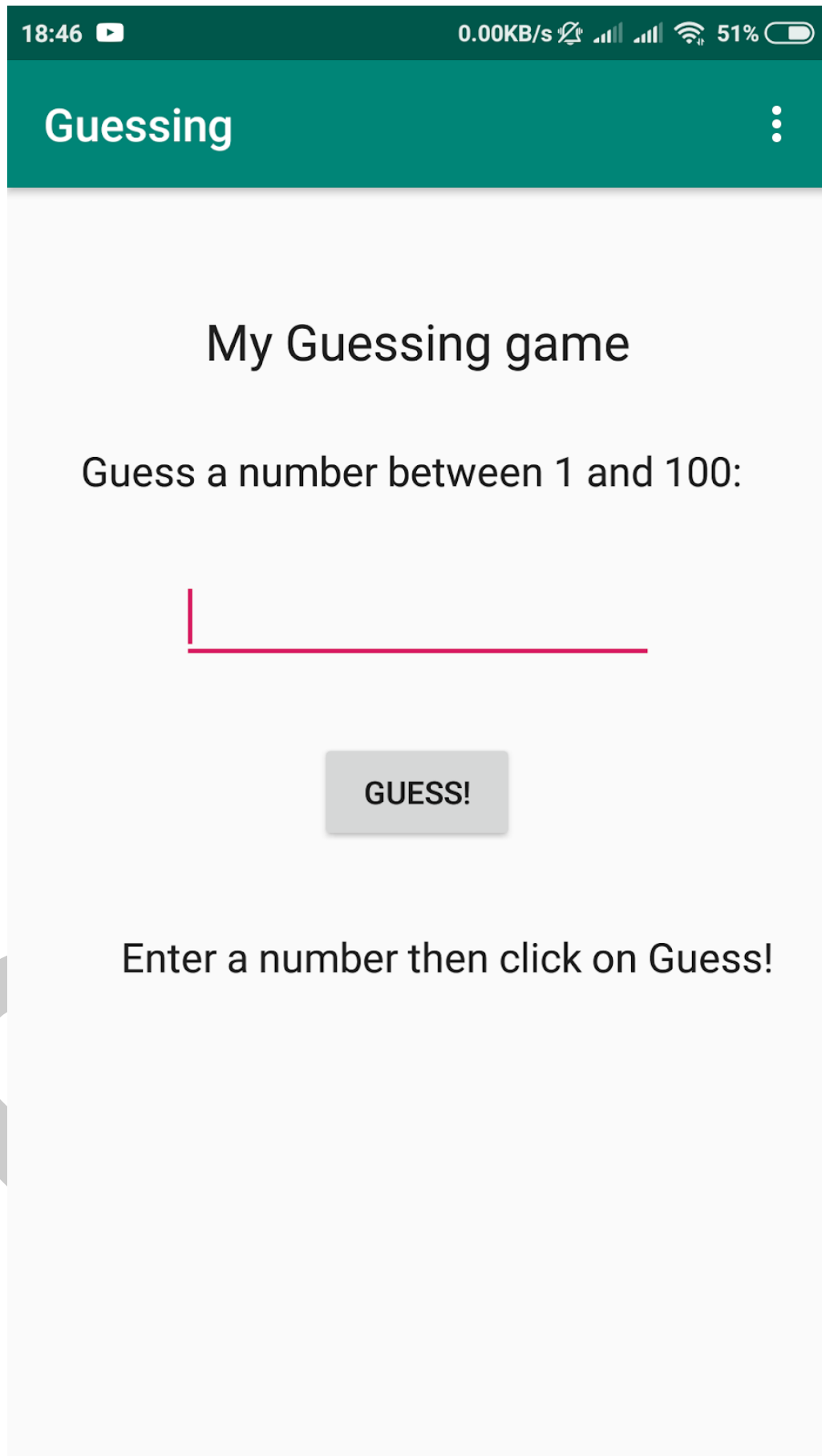
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
```

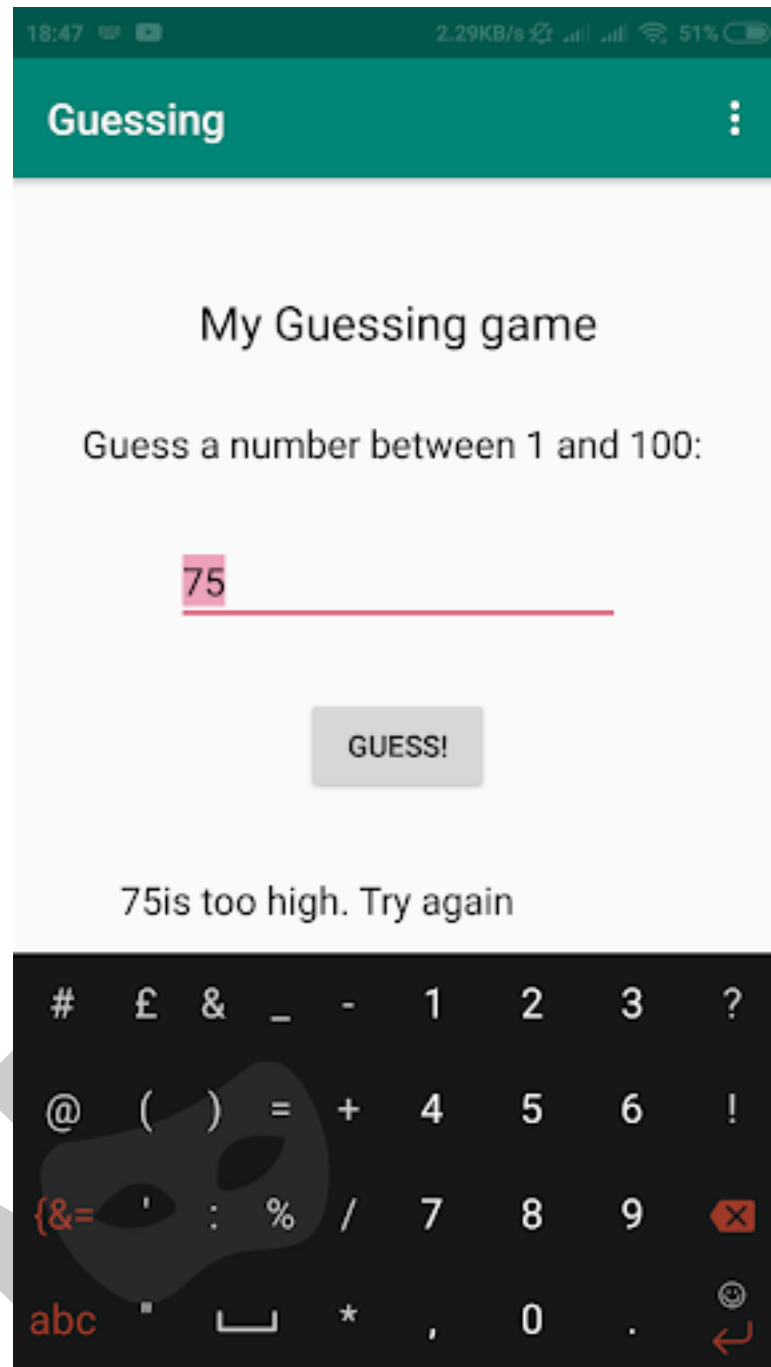
```
private TextView lblOutput;
private int theNumber;
public void checkGuess() {
    String guessText = txtGuess.getText().toString();
    String message = "";
    try {
        int guess = Integer.parseInt(guessText);
        if (guess < theNumber)
            message = guess + " is too low. Try again.";
        else if (guess > theNumber)
            message = guess + " is too high. Try again.";
        else {
            message = guess +
                " is correct. You win! Let's play again!";
            newGame();
        }
    } catch (Exception e) {
        message = "Enter a whole number between 1 and 100.";
    } finally {
        lblOutput.setText(message);
        txtGuess.requestFocus();
        txtGuess.selectAll();
    }
}

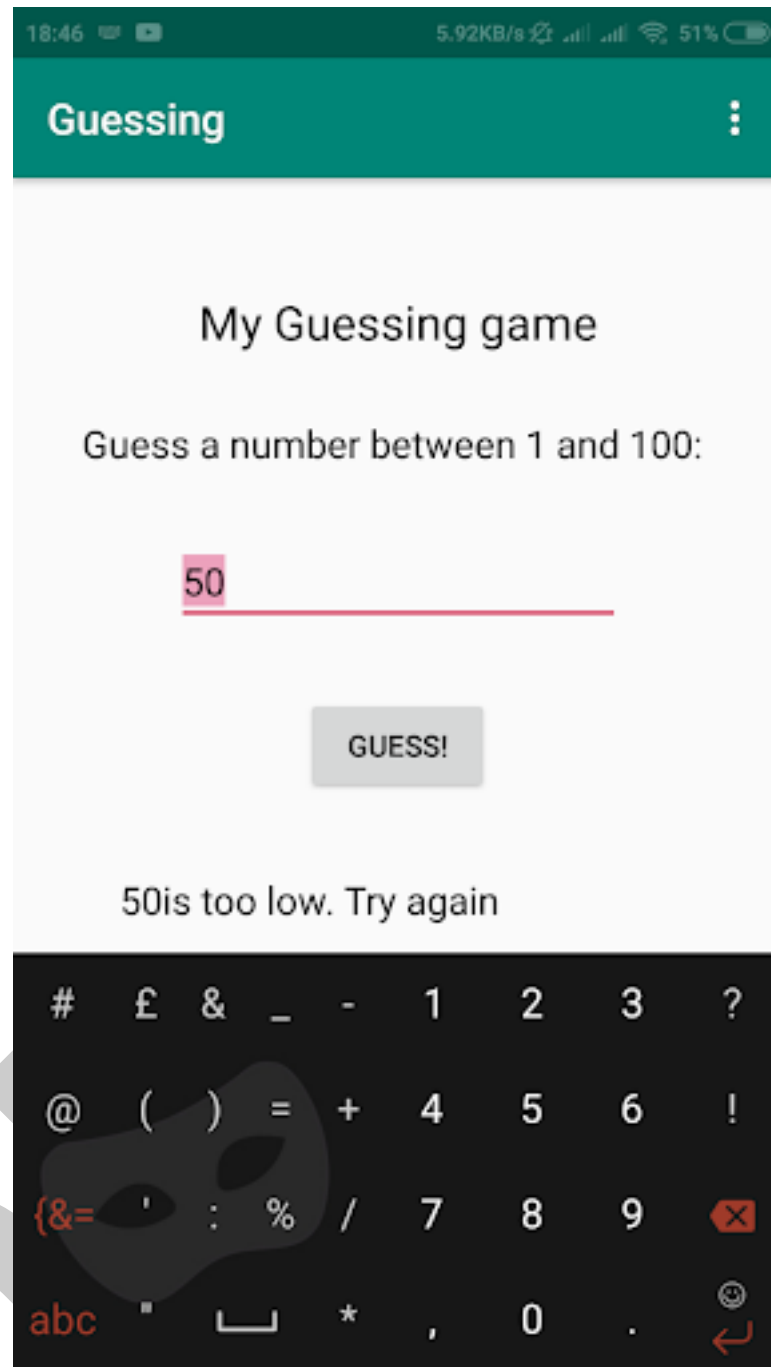
public void newGame() {
    theNumber = (int) (Math.random() * 100 + 1);
}

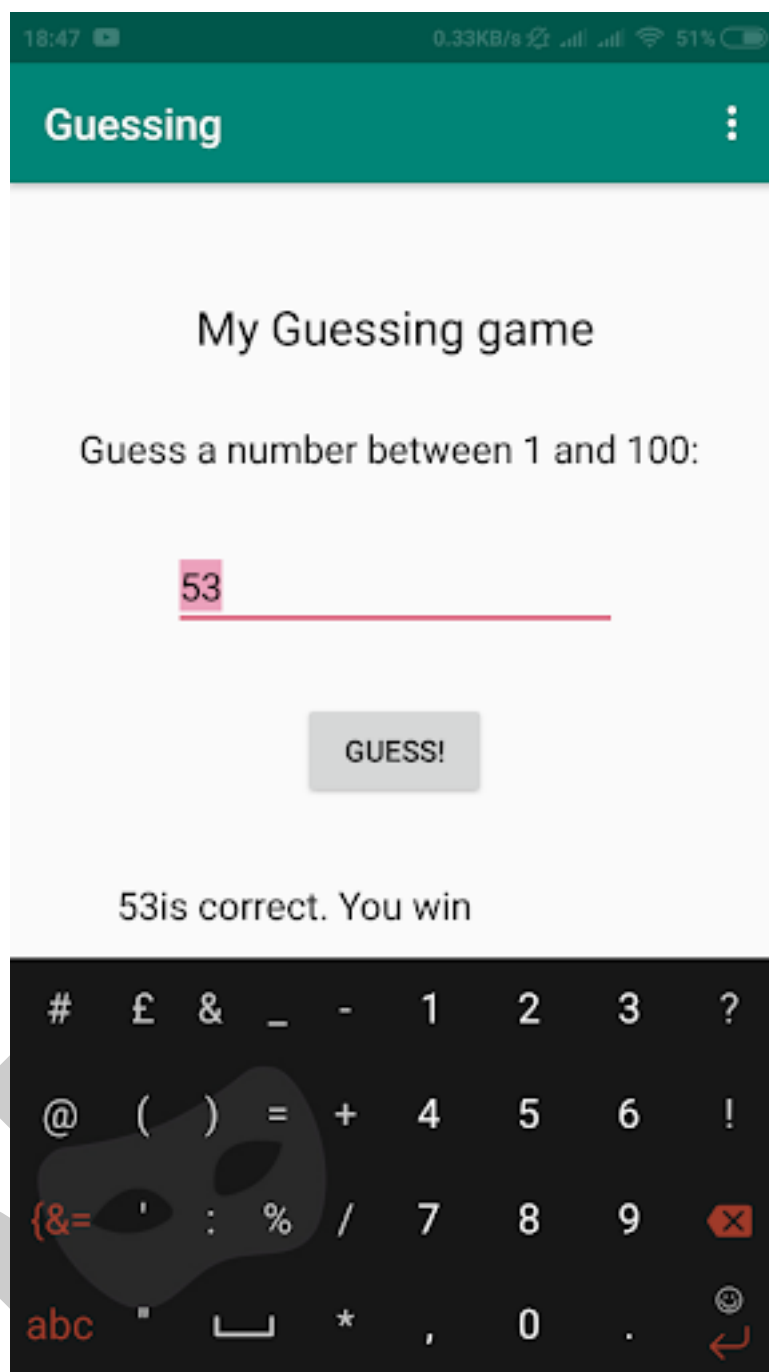
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
    newGame();
    btnGuess.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            checkGuess();
        }
    });
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
}
}
```

Pár képen látható a program működése.







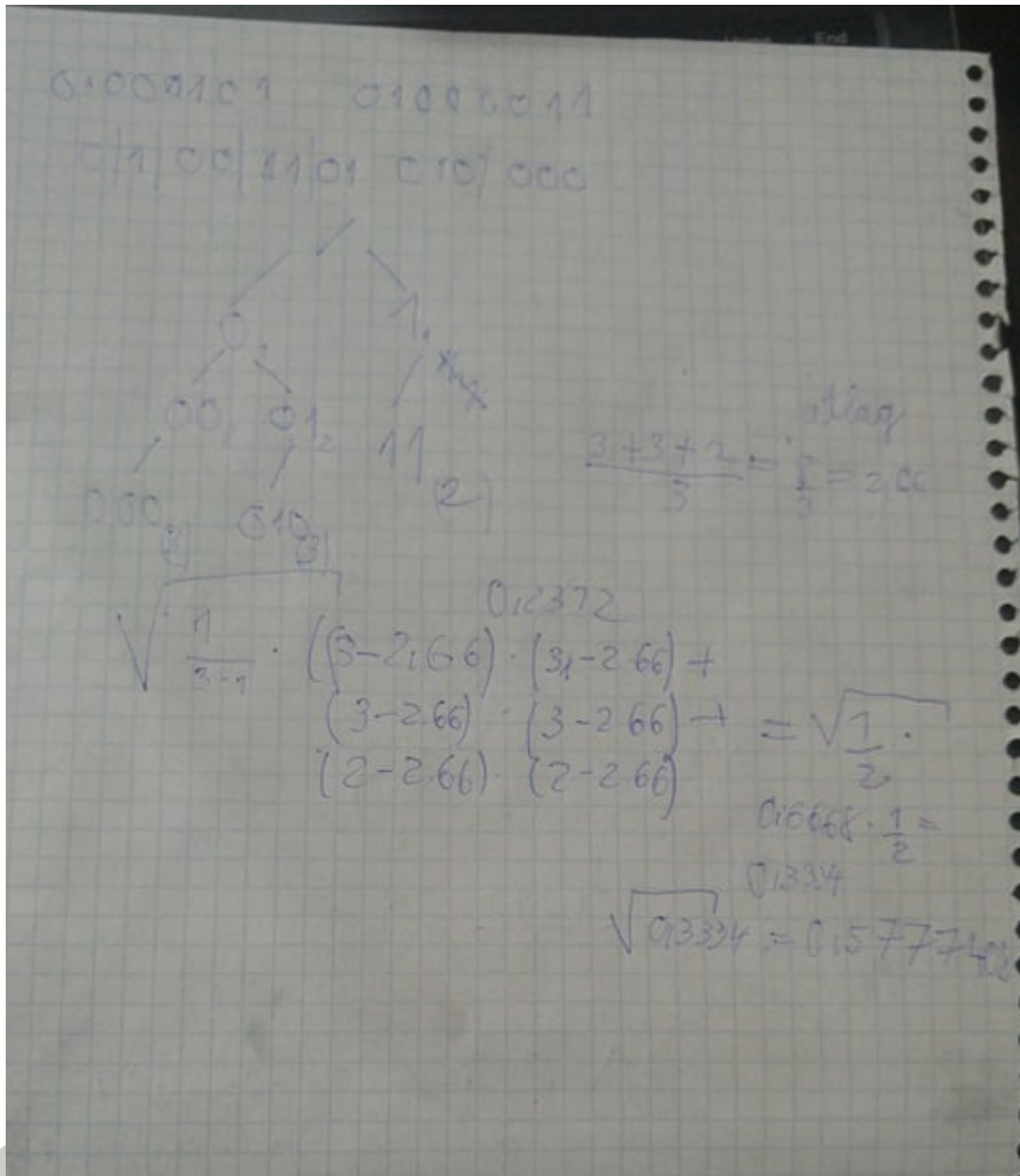


### 8.3. Junit Test

Ebben a feladatban el kellett készítenünk a binfának a számításait. Beleértve a fa mélységét, átlagát és a szórását. Kellett egy kis idő, hogy megértsem a binfának algoritmusát, de miután ez megvolt teljesen világossá vált minden ebben a témában.

```
01001101 01000011
```

Ez a kódsorozat MC -nek felel meg bináris alakban. Ahogy a feladat megkövetelte, először lapon oldottam meg. Itt látható képen.



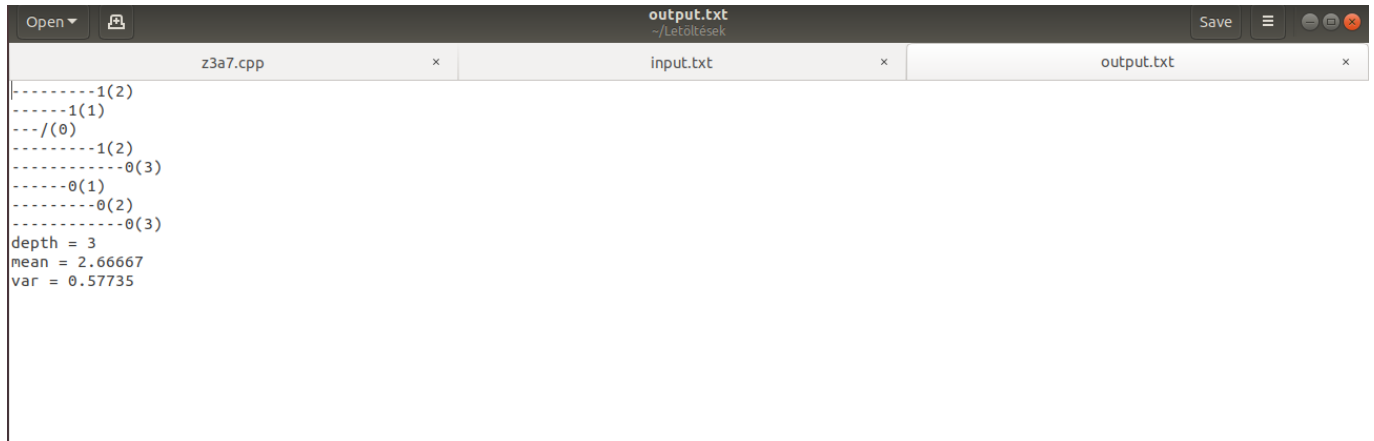
Amint látható a bináris sorban szétválasztom azokat az elemeket amelyek eddig nem voltak jelen. Ilyen a 0 az 1 és így tovább. Amelyek már megjelentek azokat, már nem veszem figyelembe, olvasom tovább a többi elemmel együtt. Vannak olyan eshetőségek, hogy például a 11 a végén már nem is kell figyelembe venni, hiszen már ezelőtt már ki volt emelve.

Ezután az elemeket egy fa struktúrában felírtam. Itt látszódnak az elágazások és ebből lehet megítélni hogy mi is a fa mélysége. A fa mélységének az értékét pedig azt adja, hogy 0-nak vagy 1-nek melyik a legutolsó eleme. Itt jelen esetben a 000 és a 010 a legmélyebben szereplő elem.

Átlagot úgy kapjuk meg, hogy az elágazások legmélyebb elemeinek a mélység értékeit összeadjuk és ahány elem van annyival osztjuk ezt az adott értéket. Itt a 000-nak és a 010-nak 3 a mélysége és emelet van még a 11 amelynek a mélysége a 2. Ezeknek az elemeknek az értékeit összeadjuk és elosztjuk 3-al jelen esetben. Így kapjuk meg a fának az átlagát, ami itt  $\frac{8}{3}$  vagy 2.66.

Ezután megnézzük a szórását. A szórás már egy jóval komplikáltabb számítás, amelyet igazából nem szeretnék nagyon részletezni. Annyit kell tudni a fa elágazások mélységét ki kell vonni az átlagból és ezeknek az értékeit vesszük a négyzetét. Ezek után a szorozzuk az  $\frac{1}{\text{mélység}-1}$  értékkel, ami itt jelen esetben

1/2. Ezzel a számmal szorozzuk az eddigi négyzeteknek összegeit. Majd ezek után ennek vesszük a négyzetgyökét. Amint látható ez már egy sokkal bonyolultabb számítási módszer, de az érték nagyjából teljesen megegyezett. Gondolom azért van egy kisebb eltérés mert én szimplán 2,66-al számoltam, míg a gép sokkal több 6-al számolt(itt 2.666667 körüli számra gondolok).



Ezen a képen látható az ellenőrzés a z3a7.cpp-vel a Tanár Úr binfájával. Amint látható a mélység és az átlag teljesen megegyezik, míg a szórás mint említettem egy kicsit különbözik.

Ez volt a feladat elméleti része a 2.része az volt, hogy a binfának a számolási függvényeit JUnittal ellenőrizzem. A feladatot eclipseben próbáltam elkészíteni, hiszen az eclipseben vannak beépített JUnit tesztlők. Elsődleges feladat az volt, hogy a Javas binfát alkalmazzam és azt olvassa be. Miután megadtam neki a Binfás fájlmot, ki kellett választani, hogy milyen függvényeket és metódusokat teszteljen a JUnit. Mivel csak a szórást az átlagot és a mélységet kellett tesztelni ezért elég ezt a 3 metódust kiválasztani. Ezután automatikusan kreált egy JUnit-os a fájlt és ebben kellett egy pár dolgot átírnunk a fájlban. Íme a kód:

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

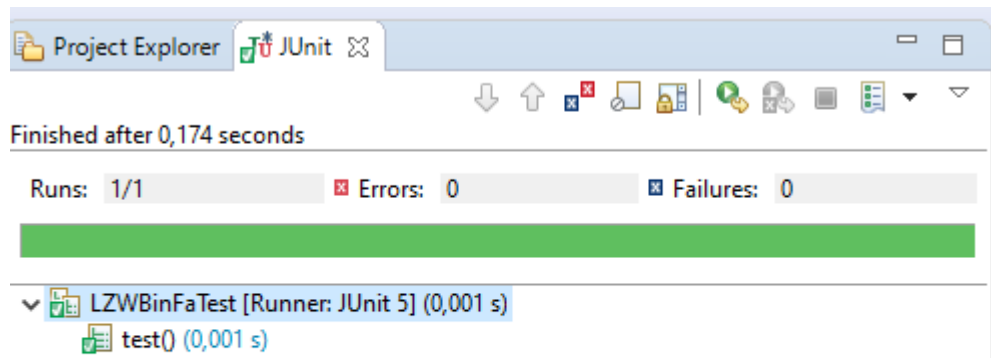
class LZWBinFaTest {
    LZWBinFa binfa = new LZWBinFa();

    @Test
    void test() {
        for (char c : "0100110101000011".toCharArray())
            binfa.egyBitFeldolg(c);

        org.junit.Assert.assertEquals(3, binfa.getMelyseg(), 0.0);
        org.junit.Assert.assertEquals(2.66667, binfa.getAtlag(), 0.001);
        org.junit.Assert.assertEquals(0.57735, binfa.getSzoras(), 0.0001);
    }
}
```

Amint látható először is példányosítottuk az LZWBinFát. Majd a teszt függvényben for ciklussal az összes 0 1 karaktert egy karaktertömbbé alakítjuk. Ezután az Assertben összevetjük az adatokat. Az assert függvényben van 3 paraméter az első érték amelyet kaptunk a 2. érték a binfa értéke a 3. pedig az eltérésnek az összegét adtuk meg. Abban az esetben ha valami hiba van akkor a JUnit test sikertelen, de amint látható a képen a testünk sikeres volt.





## 9. fejezet

# Helló, Berners-Lee!

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

A Java és a C++ magasszintű, objektumorientált programozási nyelv. Nagyon sok hasonlóság van a két nyelvben, hiszen a Java nagyon sok dolgot vett át a C++ tól. A Java viszont újabb ezért a fejlesztők sokat dolgoztak azon, hogy a Java kódok megbízhatóbbak, biztonságosabb és emelett platformfüggetlenek legyenek. Mint említettem, a két nyelv objektumorientált, de a C++ nem kötelez arra hogy osztályokba dolgozzunk, ezzel szembe a Javában csak objektumorientáltan lehet dolgozni.

A Java és a C++ szintaxisa nagyon hasonló, hiszen a Java a C és a C++ szintaxisán alapszik. Természetesen vannak apróbb különbségek. Később kifejtem...

A két nyelv mint említettem nagyon hasonlít azonban van egy nagy különbség. A memóriakezelés a C++-ban manuális míg Javában a memóriakezelést a Garbage collection "szemét gyűjtő" végzi ezt a munkát. A Javában a rendszer önmaga kezeli a memóriát és ha rendszer úgy érzi, hogy betelik a memória, akkor felszabadít memóriát. A C++-ban a programozónak kell a memóriát kezelnie. A C++-ban destruktorokat alkalmazunk ahhoz, hogy memóriát szabadítsunk fel. Természetesen mindkettőnek megvannak előnyei, hátrányai. Javában az előny az, hogy nem kell nekünk kezelni ezeket a memória felszabadításokat, amely egy nagyobb projektnél egy nagyon kényelmes funkció, azonban mi ezt nem tudjuk irányítani, ezért lehetséges az, hogy feleslegesen tud nagyobb méretű memóriát foglalni a rendszer egy kisebb programhoz, mint amennyire aktuálisan szükségünk van rá.

### 9.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. I--II.II.

Guido van Rossum 1990-ben a Pythont, amely egy általános célú magasszintű programozási nyelv. Ez a nyelv a fejlesztők számára rengeteg pozitív tulajdonsággal rendelkezik ezek közül pl:dinamikus,objektumorientált és a platformfüggetlenség. Python tulajdonképpen egy szkriptnyelv, azonban rendkívül sok csomagot és beépített eljárását tartalmaz, ezért összetett alkalmazások készítésére is alkalmas.

Nagyon sok platformokon érhető el pl: Win, Mac, Unix, S60, UIQ, Iphone. A Python egy köztes nyelv, nincs szükség fordításra és linkelésre. Az értelmező (interpreter) interaktívan is használható amelynek segítségével például egyszerűen kezelhetünk (throw-away) programokat is bizonyos funkciók kipróbálására.

A nyelv legfőbb jellemzője, hogy behúzásalapú a szintaxisa. A programban szereplő állításokat az azonos szintű behúzásokkal tudjuk csoportokba szervezni, nincs szükség kapcsos zárójelre vagy explicit kulcsszavakra(pl.begin,end). Fontos, hogy a behúzásokat egységesen kezeljük, tehát vagy mindenhol tabot, vagy egységesen szóközt használjuk.

A nyelv további sajátossága, hogy a sor végéig tart egy utasítás, nincs szükség a megszokott ';' használatára. Ha egy utasítás több sorban fér csak el, akkor ezt a sor végére '\n' (backslash) jellel lehet jelezni. Amennyiben nem zártunk be minden nyitott zárójelet akkor az utasítás folytatósorának veszi a következő sort. Az értelmező minden egyes sort úgynevezett tokenekre bont, amelyek közt tetszőleges üres whitespace karakter lehet. Tokenek fajtái:azonosító kulcsszó,operátor,literál. Kulcsszavakat itt is is alkalmazunk pl : lambda,return,while,try stb.

Pythonban minden adatot objektumok reprezentálnak. Pythonban nincs szükség a változók típusainak explicit megadására,a rendszer futási időben,automatikusan „kitalálja” a változók típusát. Az adattípusok a következők lehetnek: számok, sztringek,ennek (tuples,n-es), listák, szótárak. A számok lehetnek egészek, lebegőpontosak és komplex számok. Az egész számok lehetnek decimálisak,oktálisak.

## **II. rész**

### **Irodalomjegyzék**

DRAFT

### 9.3. Általános

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

### 9.4. C

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

### 9.5. C++

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

### 9.6. Lisp

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

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

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