

# **Univerzális programozás**

**Fekete Dávid saját programozási tankönyve  
Prog2.**

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

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Fekete Dávid

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

Copyright (C) 2019, Fekete Dávid , davidfekete51@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	Bátfai, Norbert ÁCs Fekete, Dávid	2019. december 13.	

**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
0.1.0	2019-09-19	Berners lee csokor, olvasónapló.	davidfekete
0.1.1	2019-09-26	Arroway csokor megoldása.	davidfekete

# 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]

DRAFT

# Tartalomjegyzék

<b>I. Bevezetés</b>	<b>1</b>
1. Vízió	2
1.1. Mi a programozás? . . . . .	2
1.2. Milyen doksikat olvassak el? . . . . .	2
1.3. Milyen filmeket nézzek meg? . . . . .	2
<b>II. Második felvonás</b>	<b>3</b>
2. Helló, Berners-Lee!	5
2.1. JAVA, C++ összehasonlítás: . . . . .	5
2.2. Python olvasó napló . . . . .	6
3. Helló, Arroway!	7
3.1. OO szemlélet . . . . .	7
3.2. Homokozó . . . . .	10
3.3. „Gagyi” . . . . .	10
3.4. Yoda . . . . .	13
3.5. Kódolás from scratch . . . . .	15
4. Helló, Liskov	16
4.1. Szülő-gyerek . . . . .	16
4.2. Ciklomatikus komplexitás . . . . .	18
4.3. Liskov helyettesítés sértése . . . . .	19
5. Helló, Mandelbrot!	22
5.1. Reverse engineering UML osztálydiagram . . . . .	22
5.2. Forward engineering UML osztálydiagram . . . . .	23
5.3. BPMN . . . . .	24

<b>6. Helló, Chomsky!</b>	<b>26</b>
6.1. Encoding . . . . .	26
6.2. I334d1c4 . . . . .	27
6.3. Full screen . . . . .	30
<b>7. Helló, Stroustrup!</b>	<b>33</b>
7.1. JDK osztályok . . . . .	33
7.2. Másoló-mozgató szemantika . . . . .	35
7.3. Összefoglaló: JDK osztályok . . . . .	37
<b>8. Helló, Gödel!</b>	<b>38</b>
8.1. STL map érték szerinti rendezése . . . . .	38
8.2. Alternatív Tabella rendezése . . . . .	39
8.3. Gengszterek . . . . .	41
<b>9. Helló,!</b>	<b>43</b>
9.1. FUTURE tevékenység editor . . . . .	43
9.2. OSM térképre rajzolása . . . . .	44
9.3. OOCWC Boost ASIO hálózatkezelése . . . . .	45
<b>10. Helló, Lauda!</b>	<b>46</b>
10.1. Port scan . . . . .	46
10.2. Android Játék . . . . .	47
10.3. Junit teszt . . . . .	48
<b>11. Helló, Calvin!</b>	<b>49</b>
11.1. MNIST . . . . .	49
11.2. Android telefonra a TF objektum detektálója . . . . .	50
11.3. . . . .	50
11.4. . . . .	50
<b>III. Irodalomjegyzék</b>	<b>51</b>
11.5. Általános . . . . .	52
11.6. C . . . . .	52
11.7. C++ . . . . .	52
11.8. Lisp . . . . .	52

# Előszó

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

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

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

## Hogyan forgasd

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

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben 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 másit is) példával.

## Hogyan nyomjuk?

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

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

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

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



#### A DocBook XML 5.1 új neked?

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

## I. rész

### Bevezetés

DRAFT

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

### 1.2. Milyen doksikat olvassak el?

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

### 1.3. Milyen filmeket nézzek meg?

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

**II. rész**

**Második felvonás**

DRAFT

**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

## 2. fejezet

# Helló, Berners-Lee!

### 2.1. JAVA, C++ összehasonlítás:

Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

C++: Benedek Zoltán, Levendovszky Tíhamér Szoftverfejlesztés C++ nyelven

Osztályozásnak nevezük azt a folyamatot, amelynek során a hasonló objektumokat közös csoportokba az-fafa az osztályokba soroljuk. Az objektumorientált programokban közös tervezésre ad lehetőséget, hogy sok objektum hasonló jellemzőkkel rendelkezik Nagy előnyt jelent az, ha sok hasonló objektumot közös "terv- rajz" alapján tudunk elkészíteni. Ezeket az ún. tervrajzokat hívjuk osztályoknak. Az osztály bizonyos fajta objektumok közös változót és metódusait írja le. Az ostályok definiálhatnak példányválto-zokat, osztály- változókat és osztálymetódusokat is. Az osztályváltozók az összes objektumpéldány számára megosztott információkat tartalmaznak. Ha osztályváltozókat alkalmatunk akkor feleslegesek lesznek a példányválto- zók. Az osztályok legnagyobb előnye az újrafeljhasználhatóság. Az objektum változókból és kapcsolódó metódusokból felépített egység. Az objektum tulajdonságait célszerű elrejteni tehát nem publikusként ke- zálni és csak a metódusokon keresztül befolyásolni. Az objektumok használatának legnagyobb előnye a modularitás és az információelrejtés. Modularitás: Az objektum forráskódja független marad más objektumok forráskódjától és ennek köszönhetően könnyen tud illeszkedni a rendszer különböző részei-hez. Információ elrejtés: Az objektum a publikus interfészén kommunikál a többi objektum felé. Illetve gondos- kodik a saját adatairól és csak a metódusain keresztül ad változtatási lehetőséget a külső objektumoknak. A külső objektumoknak igazából nem is kell tudnia arról, hogy az objektum állapota hogyan van reprezen- tálva, csak a kívánt viselkedést kell kérnie a metódusokon keresztül. A pédányosításhoz a new operátort használjuk a Javaban. Ez egy új példányt hoz létre az osztályból és foglal helyet a memoriában. Szüksé- günk van egy konstruktorra is ami a hívást írja elő, ennek a neve adja majd meg, hogy melyik osztályból kell létrehozni az új példányt és inicializálja az új objektumot. A new operátor egy hivatkozást ad vissza a létrehozott objektumra. Gyakran ezt a hivatkozást hozzárendeljük egy változóhoz. Ha a hivatkozás nincs hozzárendelve változóhoz, az objektumot nem lehet majd elérni, miután a new operátort tartalmazó utasítás végrehajtódott. Az ilyen objektumot névtelen objektumnak is szoktuk nevezni. A java fordító egy bájtkód- nak nevezett formátumra fordítja le a forráskódot amit a jvm önálló interpreterként fog érzékelni. Előnyös biztosnági szempontból de lassú, ezért minden jó jvm próbálja növelni a sebességet. A kódot for-dítás előtt platformfüggő gépi kódra alakítja át. A nyelv szintaxisa c, c++ ból fejlődött ki, ez a szerkezetben jelenik meg de a java el is tér tőlük vagyis a hasonlóság nem egyenlő az azonossággal. C és c++-al szemben nincs alapértelmezett visszatérési érték, mindig meg kell azt adni, váltózókhöz "="-el lehet értéket rendelni kez- deti értékadás után nincs definiálva az érték. Még egy különbség, hogy név túlterhelés ugyan van a

Javaban is viszont operátor túlterhelés nincs benne. Illetve számos c++ kifejezés, utasítás szintaktikailag helyes Javaban is sőt sokszor a jelenetése is megegyezik. A java mint nyelv szűkebb a c++nál, de az osztálykönyvtárai miatt szélesebb az alkalmazhatósági területe. Támogatja például a GUI programozást, network programozást vagy éppen a perzisztenciát is. C++-ban is lehet ezeket csinálni de van amelyikhez külső könyvtárak segítségét kell igénybe vennünk. Valamint lehet benne forrás szinten hordozható programokat írni de a szerkesztett bináris kód már nem hordozható, mert a lefordított kód tartalmazza a helyi oprendszerre és hardverre vonatkozó feltételezéseket. A Java egyik fő célja és egyben egyik legnagyobb különbsége a c++-hoz képest az, hogy a kód teljes mértékben hordozható. Emiatt a Java szigorúbb előírásokat szab a típusok méretére, belső szerkezetére, a kifejezések kiértékelésére és a kivételek kiváltásának ellenőrzésére. A statikus változók inicializálása is futási időben történik Javaban valamint sokkal kevesebb dolgot bíz az implementációra mint a c vagy a c++. Ezt azért teszi, hogy maga a kód amely hordozható ne függjen annyira a platformtól és az implementációtól. A Java nyelv nagyon odafigyel arra, hogy a kód a lehető leg pontosabban forduljon le és működjön. Ezért az ellenőrzés során kitér olyan dolgokra is amire a c++ és a c nem. Ilyen például a lokális változók ellenőrzése, pontosabban annak ellenőrzése, hogy kapnak-e értéket.

## 2.2. Python olvasó napló

Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípusfejlesztés Python és Java nyelven (35-51 oldal)

A phyton programozási nyelv a C++-tól, c-től és a Javától eltérő módon arra lett inkább tervezve hogy ne a futási sebességet tegye előtérbe, hanem inkább a programozót segítse azzal, hogy könnyebben olvasható. Maga a Phyton egy nagyon magas szintű programozási nyelv melyet 1989 és 1991 között alkottak meg. Egy objektumorientált interpreteres nyelv (tehát rögtön futtatható, nincs különbség a forrás és a tárgykód között). Ahogy a könyv címe is sejteti az olvasóval a Phyton legelterjettebb felhasználási területe a mobilprogramozás. A nyelv legfőbb jellemzője ami megkülönbözteti az általunk már jobban ismert nyelvektől és ujdonság lehet a számunkra, hogy a szntaxisa behúzás alapú. Ez azt jelenti h az állításokat azonos szintű behúzásokkal tudjuk csoportosítani. Nem kell kapcsos zárójeleket és kulcsszavakat mint például a begin és az end használatba vennünk ehhez a feladathoz. Nagyon fontos, hogy az első utasítás a szkriptben nem lehet behúzás illetve ezeket egységesen kell kezelnünk. Továbbá az utasítások csak a sor végéig tartanak nem kell ezeket lezárnai. Ha mégsem férne egy utasítás egy sorba akkor '/'-el tudjuk ezt folytatni a következő sorba.

## 3. fejezet

# Helló, Arroway!

### 3.1. OO szemlélet

módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1\\_5.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf) (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

JAVA forras:

```
public class PolarGenerator {
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator() {
        nincsTarolt = true;
    }
    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;
                w = v1 * v1 + v2 * v2;
            } while(w > 1);
            double r = Math.sqrt(-2 * Math.log(w) / w);
            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;
            return r * v1;
        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }
}
```

```
    }
}

public static void main(String[] args) {
PolarGenerator pg = new PolarGenerator();
for(int i = 0; i < 10; i++) {
System.out.println(pg.kovetkezo());
}
}
```

The screenshot shows an IDE interface with two tabs: `PolarGenerator.java` and `PolarGenerator`. The `PolarGenerator.java` tab displays the following Java code:

```
1  public class PolarGenerator {
2      boolean nincsTarolt = true;
3      double tarolt;
4      public PolarGenerator() {
5          nincsTarolt = true;
6      }
7      public double kovetkezo() {
8          if(nincsTarolt) {
9              double u1, u2, v1, v2, w;
10             do {
11                 u1 = Math.random();
12                 u2 = Math.random();
13                 v1 = 2 * u1 - 1;
14                 v2 = 2 * u2 - 1;
15                 w = v1 * v1 + v2 * v2;
16             } while(w > 1);
17             double r = Math.sqrt(-2 * Math.log(w) / w);
18             tarolt = r * v2;
19             nincsTarolt = !nincsTarolt;
20             return r * v1;
21         } else {
22             nincsTarolt = !nincsTarolt;
23             return tarolt;
24         }
25     }
26 }
```

The `PolarGenerator` tab shows the output of the Java application, which is a sequence of random numbers generated by the `kovetkezo()` method:

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
-2.0424536265181845
0.05296707421087484
0.796903445383798
0.15118134693433039
-0.6250452487773279
-0.4346800843293371
-0.5787399242026225
1.0172412827190924
1.5417401377874569
0.5052621983341685

Process finished with exit code 0
```

C++ forras:

```
#ifndef POLARGEN__H
#define POLARGEN__H
```

```
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>

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

#endif
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;
    }
}
int
main (int argc, char **argv) {
    PolarGen pg;
    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
    return 0;
}
```

Az objektumorientált programozás (OOP) olyan módszert nyújt a programozók számára, amely lehetővé

teszi a programok bonyolultságának csökkentését, a megbízhatóság és a hatékonyság növelését. Objektumokból, tehát a valós világ elemeinek programozási modelljeiből épít fel a programot. A C++ és a Java is objektumorientált programozási nyelv.

Remek OO bevezető példa lehet egy polártranszformációs normális generátor megírása C++-ban és Java-ban. A módosított polármódszeres algoritmus matematikai háttere a feladatmegoldás szempontjából lényegtelen, fontos viszont az a tény, hogy egy számítási lépés két normális eloszlású számot állít elő, tehát elég az előző lépés másik számát visszaadnunk.

A C++ megoldásban használjuk a scope operátort, amely lehetővé teszi, hogy hozzáférjünk az std névtérhez. Ennek köszönhetően tudunk random számot visszaadni, gyököt vonni, illetve logaritmizálni. Kiiratásnál és sortörésnél is hasznos. A nincsTarolt változóval jelöljük azt, hogy páros vagy páratlan lépésekben hívtuk-e meg a kovetkezo() függvényt. Ha értéke igaz, akkor tárolt lebegőpontos változóban van a visszaadandó szám.

A feladat feladata az, hogy rámutasson az objektum orientált programozás előnyeire, amire eddig konkrétan nem tértünk ki a könyvben. A program megvalósítása után azt kell látnunk, hogy nekünk, programozóknak a matematikai háttérrel alig kell foglalkoznunk, és mégis viszonylag komplex problémák megoldására vagyunk képesek programok segítségével. A kód alább látható.

A polártranszformációs normális generátor egy pszeudórandomszámok generálására kitalált algoritmus, amelynek magas effektivitást tulajdonítanak a matematikusok, hiszen nem egy, hanem kettő pszeudórandom szám jön létre egy lefutás során, hanem kettő, így minden páros lefutásnál elegendő az előzőleg generált számokból a másodikat visszaadni.

## 3.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciakat kell kiirtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját! 1

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 3.3. „Gagyi”

Az ismert formális „

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

” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciaja) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására 3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

Forras:

```
import java.util.Scanner;
public class gagyi {
    public static void main(String[] args) {
        Scanner sc;
        sc = new Scanner(System.in);
        Integer x= sc.nextInt();
        Integer y= sc.nextInt();

        System.out.println(x);
        System.out.println(y);

        while (x <= y && x >= y && y != x)  {
            ;
        }
    }
}
```

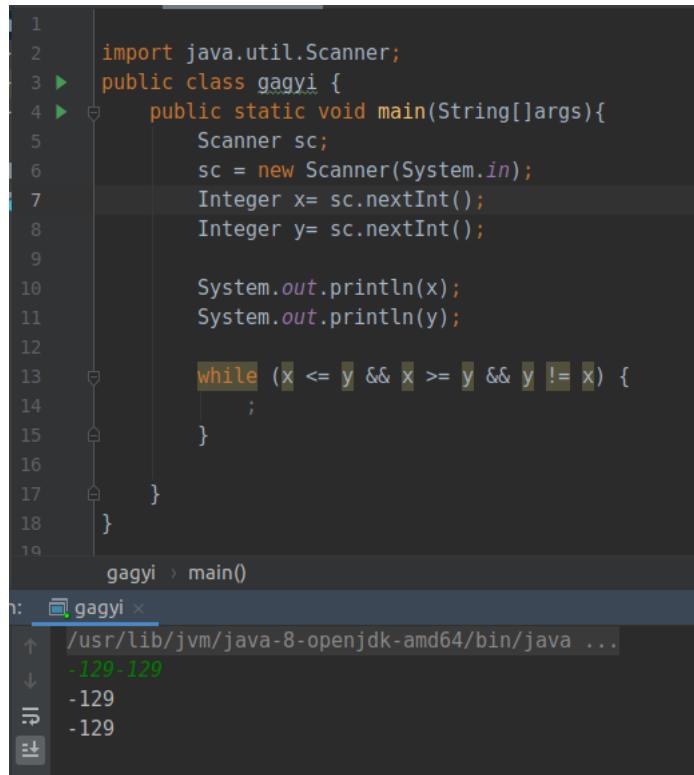
A feladat az, hogy választ adjunk a „

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

” tesztkérdéstípusra. A kérdésre a választ az Ineteger.java forrásban találjuk.

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

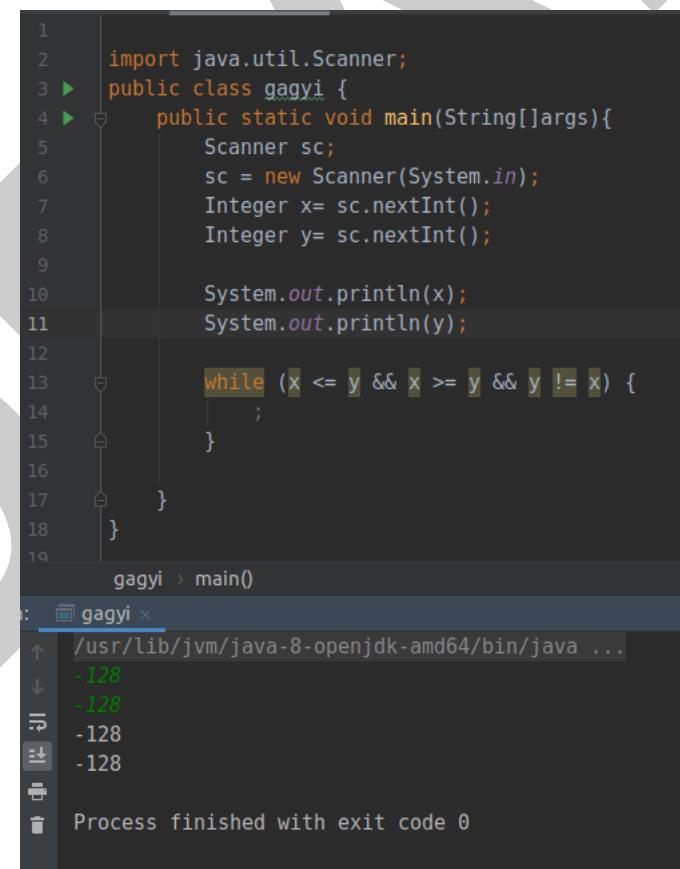
Az IntegerCache low értéke a -128, a high értéke pedig 127. Azaz csak abban az esetben lesz ugyanaz az objektum kiosztva minden a két értéknek, ha az -128 és 127 közé esik, vagy ezzel egyenlő. Ellenkező esetben a feltétel valamelyik ága meghíjusul, így a „return new Integer(i);” sor fog lefutni, vagyis különböző című objektumokat rendel majd az értékekhez. Írunk java programot, amivel ezt szemléltetni tudjuk.



```
1 import java.util.Scanner;
2 public class gagyi {
3     public static void main(String[]args){
4         Scanner sc;
5         sc = new Scanner(System.in);
6         Integer x= sc.nextInt();
7         Integer y= sc.nextInt();
8
9         System.out.println(x);
10        System.out.println(y);
11
12        while (x <= y && x >= y && y != x) {
13            ;
14        }
15    }
16 }
17
18 }
```

The code above is a Java program named `gagyi`. It uses a `Scanner` to read two integers from standard input. It then prints them out. A `while` loop is used to check if `x` is equal to `y`. Since the condition `x <= y && x >= y` is always true for any integer `x`, the loop will never exit. This results in an infinite loop.

Ahogy látjuk -129es értékekre a while feltétele teljesül, vagyis végtelen ciklus jön létre.



```
1 import java.util.Scanner;
2 public class gagyi {
3     public static void main(String[]args){
4         Scanner sc;
5         sc = new Scanner(System.in);
6         Integer x= sc.nextInt();
7         Integer y= sc.nextInt();
8
9         System.out.println(x);
10        System.out.println(y);
11
12        while (x <= y && x >= y && y != x) {
13            ;
14        }
15    }
16 }
17
18 }
```

The code above is identical to the one in the previous screenshot. When run with inputs of `-129` and `-129`, the output shows the two numbers being printed repeatedly, indicating an infinite loop.

-128as értékre pedig a while feltétele nem teljesül, így nem jön létre vételen ciklus, és befejeződik a program.

### 3.4. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-leáll, ha nem követjük a Yoda conditions-t! [https://en.wikipedia.org/wiki/Yoda\\_conditions](https://en.wikipedia.org/wiki/Yoda_conditions)

A programozásban a Yoda Conditons egy programozói stílus, ahol egy feltétel két része meg van cserélve egymással. A Yoda Conditions a konstans részét a feltétel bal oldalára helyezi. Ennek a programozási stílusnak a neve a Star Wars című filmből ered, amiben egy Yoda nevű karakter nem szabványos nyelvtannal beszél.

Két féle hibától is megóv a Yoda conditions: 1. ha összehasonlítás helyett értékadás történik 2. null értékű string összehasonlításnál

Irjuk meg a programot az 1. esetre, és szándékosan kövessük el a hibát:

The screenshot shows a Java code editor and a terminal window. The code editor displays a file named 'yoda.java' with the following content:

```
1  public class yoda {
2      public static void main(String[] args){
3          /*
4              String a = null;
5              String b = "asd";
6              if (b.equals(a)){
7                  System.out.println("OK");
8              }
9              */
10             Boolean c = true;
11             if (c = null) {
12                 System.out.println("OK");
13             }
14         }
15     }
```

The terminal window below shows the output of running the program:

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Exception in thread "main" java.lang.NullPointerException
at yoda.main(yoda.java:12)
Process finished with exit code 1
```

Tehát direkt elkövetjük a hibát: összehasonlítás helyett (==) értékadás törénik (=), láthatjuk, hogy a program lefordul és futtatásnál java.lang.NullPointerException hibát jelez.

Ha pedig követjük a Yava conditionst , vagyis felcseréljük a feltétel két részét, a program már a fordításnál jelzi a hibát (és ez nyilván előnyösebb, mivel a fordításnál kapott hiba alapján könnyebben megtaláljuk a programkódunkban a hibát):

The screenshot shows an IDE interface with a code editor and a terminal window.

**Code Editor:**

```
1
2  public class yoda {
3      public static void main(String[]args){
4          /*
5              String a = null;
6              String b = "asd";
7              if (b.equals(a)){
8                  System.out.println("OK");
9              }
10             */
11             Boolean c = true;
12             if (null = c) {
13                 System.out.println("OK");
14             }
15         }
16     }
```

**Terminal (Build Log):**

```
ges: Build ×
  Information: java: Errors occurred while compiling module 'Yoda'
  Information: javac 1.8.0_222 was used to compile java sources
  Information: 26. 9. 2019 18:43 - Build completed with 1 error and 0 warnings in 2 s 146 ms
  /home/david/Yoda/src/yoda.java
    Error:(12, 13) java: unexpected type
      required: variable
      found:   value
```

2. eset, kövessük a Yoda conditonst: (Tulajdonképpen a 2. eset is az elsőnek egy változata: itt egy függvényhívással történik a feltétel ellenőrzése)

The screenshot shows an IDE interface with a code editor and a terminal window.

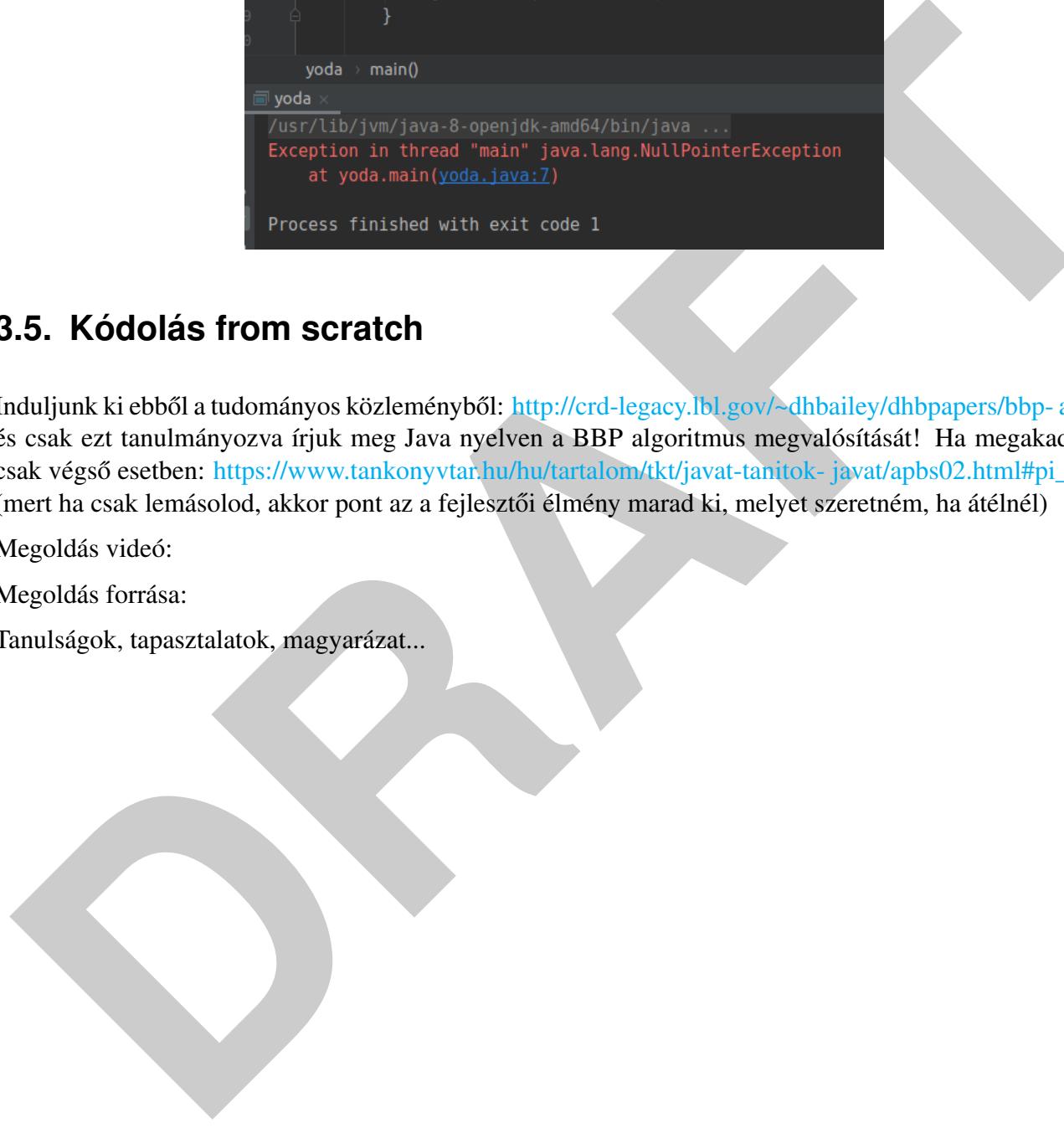
**Code Editor:**

```
▶  public class yoda {
▶      public static void main(String[]args){
>
>          String a = null;
>          String b = "asd";
>          if (b.equals(a)){
>              System.out.println("OK");
>          }
>      }
>  }
```

**Terminal:**

```
yoda > main()
yoda ×
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Process finished with exit code 0
```

Yoda conditions követés nélkül java.lang.NullPointerException hiba:



```
1  public class yoda {  
2      public static void main(String[] args){  
3          String a = null;  
4          String b = "asd";  
5          if (a.equals(b)){  
6              System.out.println("OK");  
7          }  
8      }  
9  }
```

yoda > main()

yoda x  
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...  
Exception in thread "main" java.lang.NullPointerException  
at yoda.main(yoda.java:7)  
Process finished with exit code 1

### 3.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: [https://www.tankonyvtar.hu/hu/tartalom/tkt/javat/tanitok-javat/apbs02.html#pi\\_jegyei](https://www.tankonyvtar.hu/hu/tartalom/tkt/javat/tanitok-javat/apbs02.html#pi_jegyei) (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 4. fejezet

# Helló, Liskov

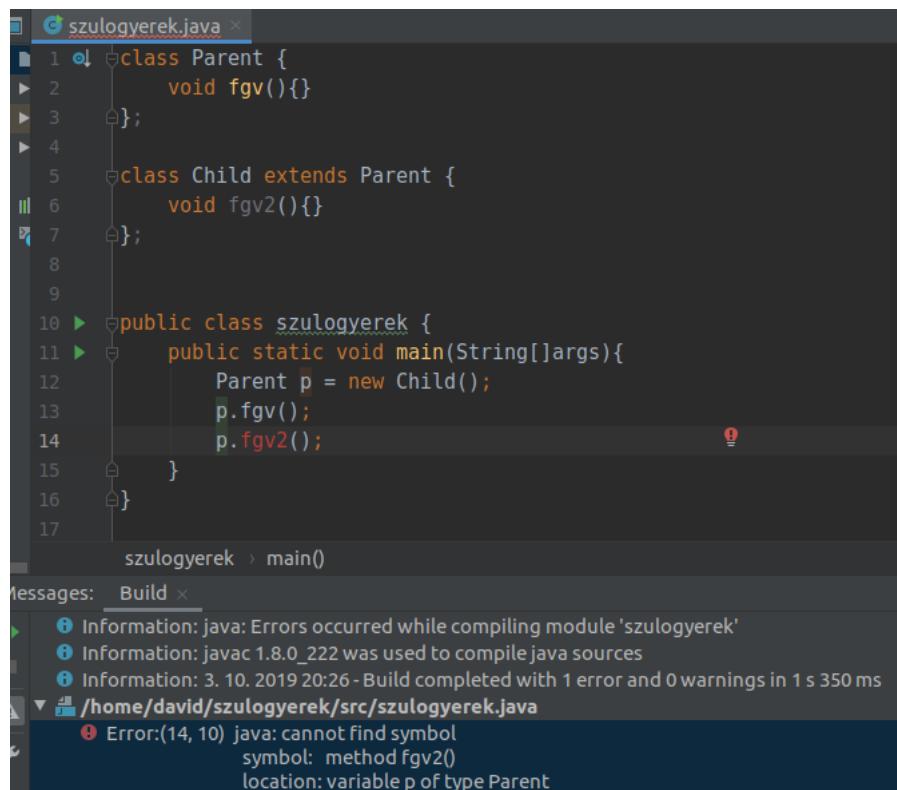
### 4.1. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf) (98. fólia)

A feladat teljesítéséhez tehát írunk Java és C++ programot, amivel bemutatjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek.

Irunk szülő-gyerek példaprogramot javaban:

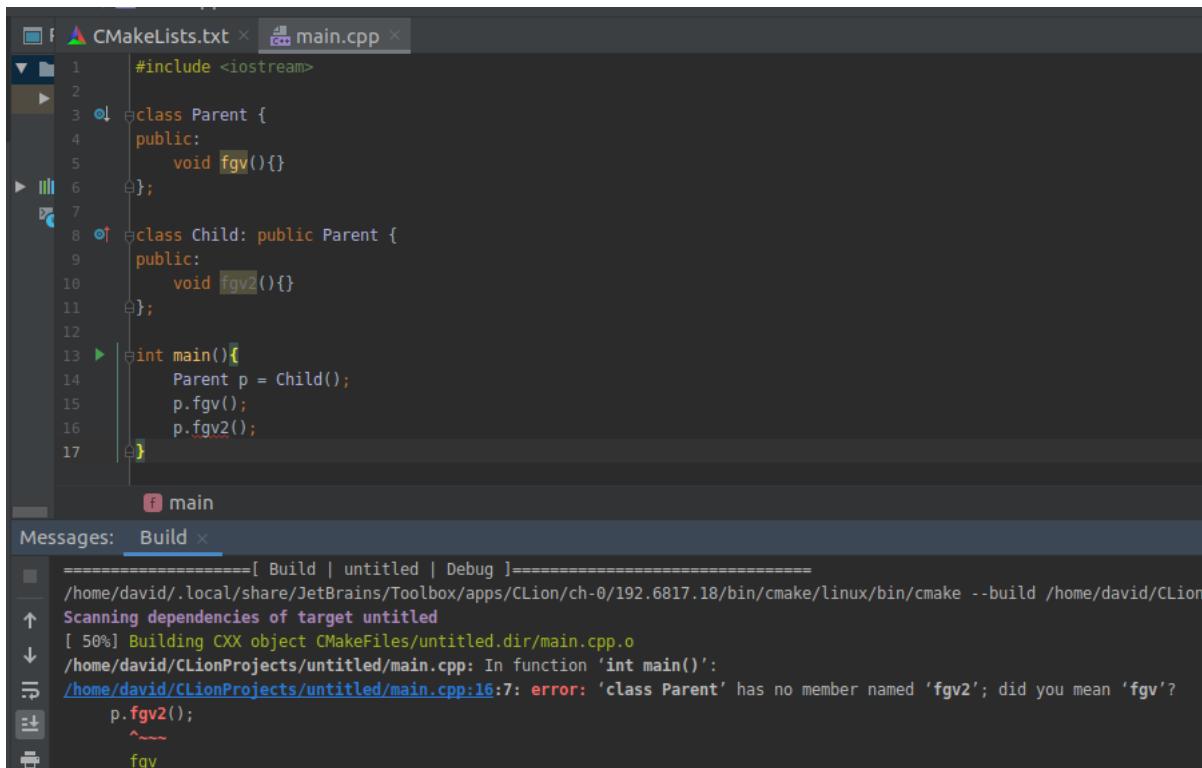
```
class Parent {  
    void fgv() {}  
};  
  
class Child extends Parent {  
    void fgv2() {}  
};  
  
public class szulogyerek {  
    public static void main(String[] args) {  
        Parent p = new Child();  
        p.fgv();  
        p.fgv2();  
    }  
}
```



A screenshot of an IDE showing a Java file named `szulogyerek.java`. The code defines a `Parent` class with a `fgv()` method, a `Child` class that extends `Parent` and overrides `fgv()`, and a `szulogyerek` class with a `main` method that creates a `Child` object and calls its `fgv()` and `fgv2()` methods. The `fgv2()` call in the `main` method is underlined with a red squiggle, indicating an error. The messages panel shows a build error at line 14, column 10: "Error:(14, 10) java: cannot find symbol symbol: method fgv2() location: variable p of type Parent".

Ugynézett pedig c++ban is próbáljuk ki:

```
class Parent {  
    public:  
        void fgv(){}  
};  
  
class Child: public Parent {  
    public:  
        void fgv2(){}  
};  
  
int main(){  
    Parent p = Child();  
    p.fgv();  
    p.fgv2();  
}
```



```
#include <iostream>
class Parent {
public:
    void fgv(){}
};
class Child: public Parent {
public:
    void fgv2(){}
};
int main(){
    Parent p = Child();
    p.fgv();
    p.fgv2();
}
```

A fordításnál error kapunk minden esetben, mivel a Parent típusú class-on keresztül nem érjük el a Child típusú class függvényeit.

## 4.2. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_2.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf)(77-79 fóliát)!

A ciklomatikus komplexitás kiszámítása:  $C = E - N + 2*P$  ahol : C = Ciklomatikus Komplexitás E = A gráf széleinek száma. N = A gráf csomópontjainak száma. P = Azon csomópontok száma, amelyeknek kilépési pontjuk van. A következő lépésekben írunk egyszerű példaprogramkódot, aminek kiszámíthatjuk a ciklomatikus komplexitását a képlet alapján.

Példaprogram c++ban:

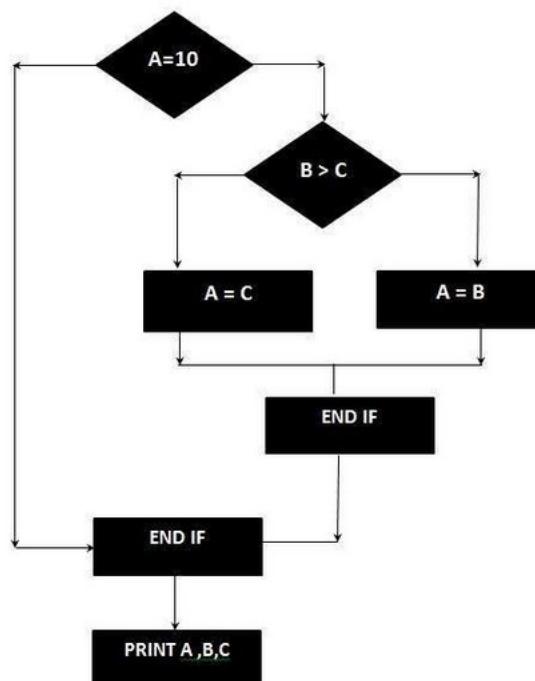
```
#include<iostream>
using namespace std;
int main() {
    int a,b,c;
    cin >> a;
    cin >> b;
    cin >> c;
    if (a == 10) {
        if (b > c) {
            a = b;
        }
        else {
```

```

    a = c;
}
}
cout<<a<<endl;
cout<<b<<endl;
cout<<c<<endl;
}

```

Ábrázoljuk a programkódot, hogy könnyebben ki tudjuk számítani a ciklomatikus komplexitást:



Ciklomatikus komplexitás = E – N +2\*P Ebben az esetben E = 8, N = 7, P = 1, tehát a példaprogram ciklomatikus komplexitása 3.

### 4.3. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_1.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf)(93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Az objektumorientált programozás öt fő tervezési elve közé tartozik az ún. Liskov-helyettesítés. Ha S osztály T osztály leszármazottja, akkor S szabadon behelyettesíthető minden olyan helyre (pl. változó), ahol T típusat várunk. Tegyük fel, hogy az Allat osztály lesz a mi példánkban a T osztály. Az S osztályaink (T osztály leszármaza- zottjai) a következők: Zebra, Oroszlan. Két osztály alkotja a P programot az LPS-ben. A programban az Zebra már nem tud vadászni, hiába lesz a leszármazott típusoknak vadász metódusa, azt a Allat allat-ra úgysem lehet hívni. Ezzel tehát a Liskov-helyettesítés elvére odafigyeltünk.

Liskovra\_figyel.java :

```
#include <iostream>
using namespace std;
class Allat {
};
class RagadozoAllat: public Allat {
public:
    void vadaszik(){
        cout << "vadaszik ..." << endl;
    }
};
class Oroszlan: public RagadozoAllat {
};
class Zebra : public Allat {
};
int main ( int argc, char **argv ) {
    Oroszlan oroszlan;
    Zebra zebra;

    // zebra.vadaszik();
    oroszlan.vadaszik();
    return 0;
}
```

Példa programunkban a Madar osztály hibásan lett definiálva, mert tartalmazza a repül() metódust, így az összes Madar-ból származtatott osztály is tartalmazni fogja a repül() funkciót. Ez alapvetően hibás, mert nem minden madár tud repülni. Hibába ütközünk, amikor a Pingvin osztályt is a madárból származtatjuk, ugyanis a pingvin nem tud repülni.

```
#include <iostream>
using namespace std;
class Allat {
};
class RagadozoAllat: public Allat {
public:
    void vadaszik(){
        cout << "vadaszik ..." << endl;
    }
};
class Oroszlan: public RagadozoAllat {
};
class Zebra : public Allat {
};
int main ( int argc, char **argv ){
    Oroszlan oroszlan;
    Zebra zebra;
    //
    zebra.vadaszik();
```

```
oroszlan.vadaszik();  
return 0;
```

A Liskov helyettesítési elv megköveteli, hogy minden osztály legyen helyettesíthető egy gyermek osztályával anélkül, hogy a program helyes működése megváltozna. Ebből a kódcsipetből kiindulva megkezdődhet az elv megsértése. Továbbra is megmaradt a T osztály, illetve az S osztályok, viszont ezúttal nem a RágadozoAllat (S) osztályban jelenik meg a "vadászik". Ezúttal tehát így a P programban is tud vadászni az állat. Sérül a Liskov-helyettesítés elve, hiszen ebben a kódban a zebra vadászik, ami lehetetlenség.

DRAFT

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. Reverse engineering UML osztálydiagram

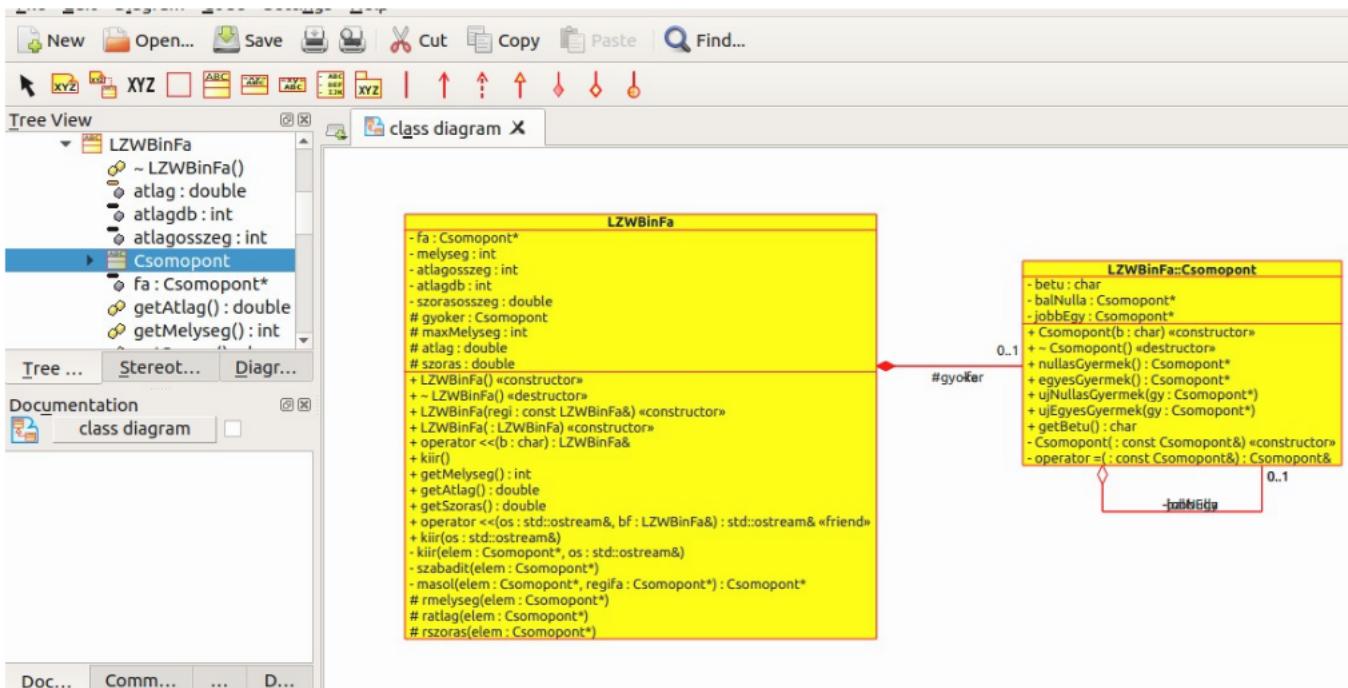
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: [https://youtu.be/Td\\_nlERIEOs](https://youtu.be/Td_nlERIEOs). <https://arato.inf.unideb.hu/batfai.norbert/UD/> (28-32 fólia)

Az UML egy egységesített modellezőnyelv, amelynek segítségével jól szemléltethetőek a fejlesztési módellek.

Kompozíció: Rész-egész kapcsolatot jelent, az egyik objektum tartalmazza vagy birtokolja a másikat.

Aggregáció: A tartalmazott a tartalmazó nélkül nem létezhet.

ULM osztálydiagram létrehozására használjuk az Umbrello nevű programot. (Miután megismerkedünk az Umbrello programmal, és saját erőből létrehozunk osztálydiagramokat, egy beépített importáló eszköz segítségével programkódóból UML osztálydiagramokat tudunk létrehozni, és fordítva: osztálydiagramokból programkódot generálhatunk. )

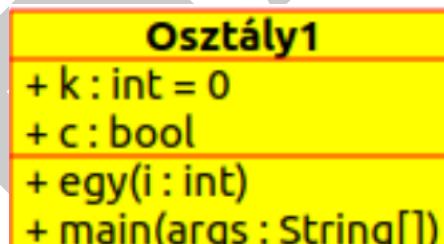


## 5.2. Forward engineering UML osztálydiagram

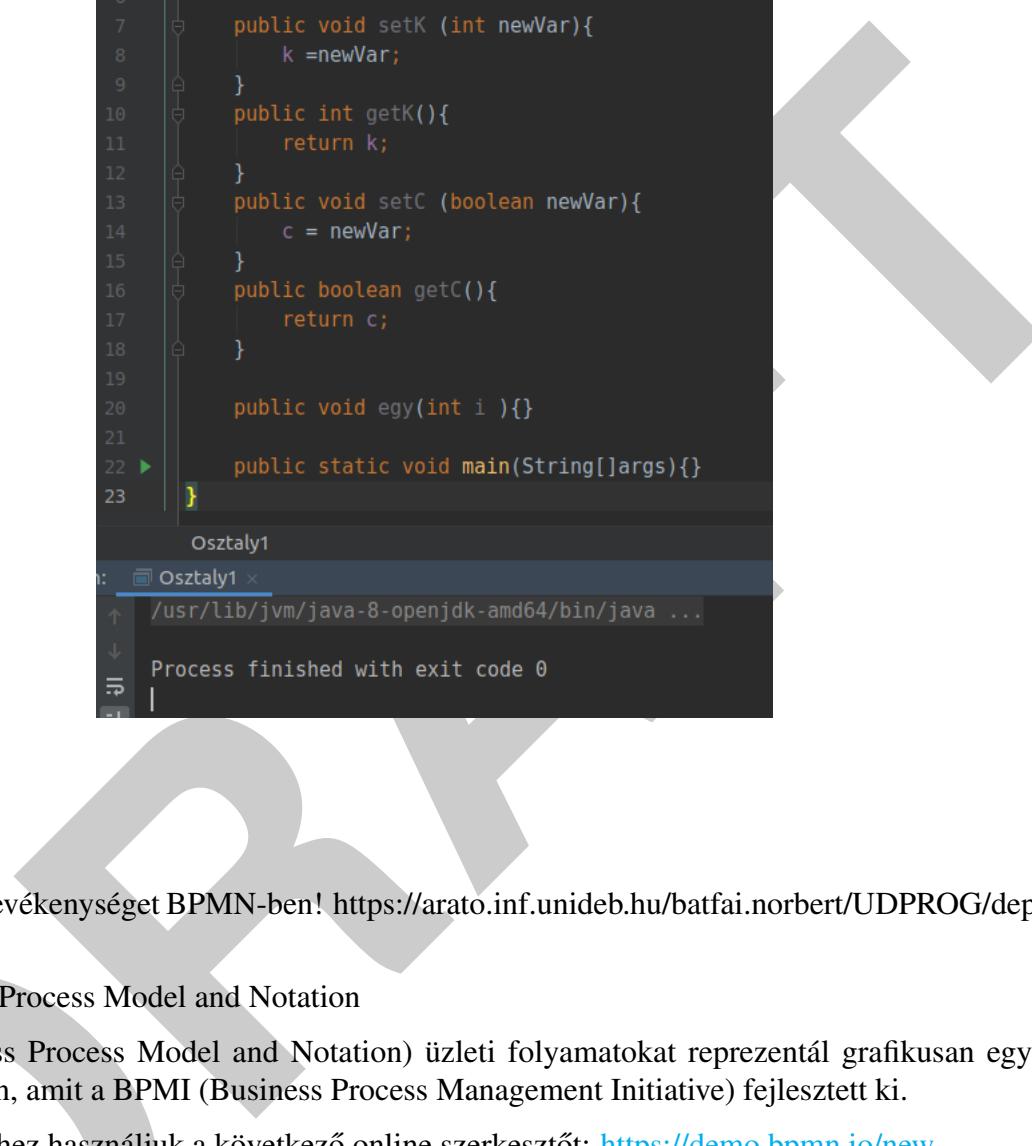
UML-ben tervezünk osztályokat és generálunk belőle forrást!

Ehhez a feladathoz is használjuk az Umbrello nevű programot.

Az umbrello lehetővé teszi, hogy osztálydiagramokhoz különböző paramétereket vegyük fel, amiknek kezdőértéket is adhatunk, vagy függvényeket adjunk hozzá, amihez paramétereket is beállíthatunk, illetve ezekhez kezdőértéket.



Ebből az egyszerű osztálydiagramból a következő java kódot kapjuk, amit fordíthatunk és futtathatunk is:



```
Osztaly1.java ×
1 ► | public class Osztaly1{
2 |     public int k = 0;
3 |     public boolean c;
4 |
5 @ |     public Osztaly1() {};
6 |
7 |     public void setK (int newVar){
8 |         k =newVar;
9 |     }
10 |    public int getK(){
11 |        return k;
12 |    }
13 |    public void setC (boolean newVar){
14 |        c = newVar;
15 |    }
16 |    public boolean getC(){
17 |        return c;
18 |    }
19 |
20 |    public void egy(int i){}
21 |
22 ► |     public static void main(String[]args){}
23 | }
```

Osztaly1

```
: Osztaly1 ×
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
Process finished with exit code 0
```

### 5.3. BPMN

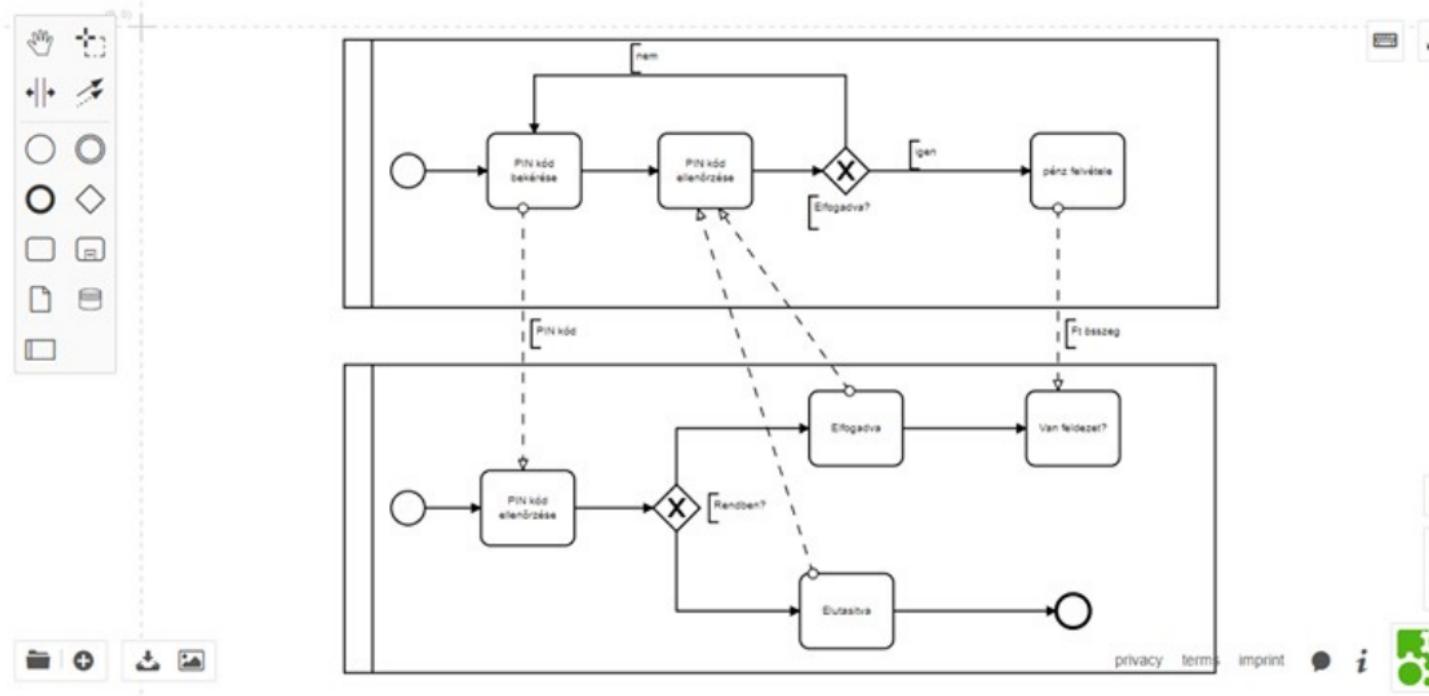
Rajzolunk le egy tevékenységet BPMN-ben! [\(34-47 fólia\)](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog)

BPMB = Business Process Model and Notation

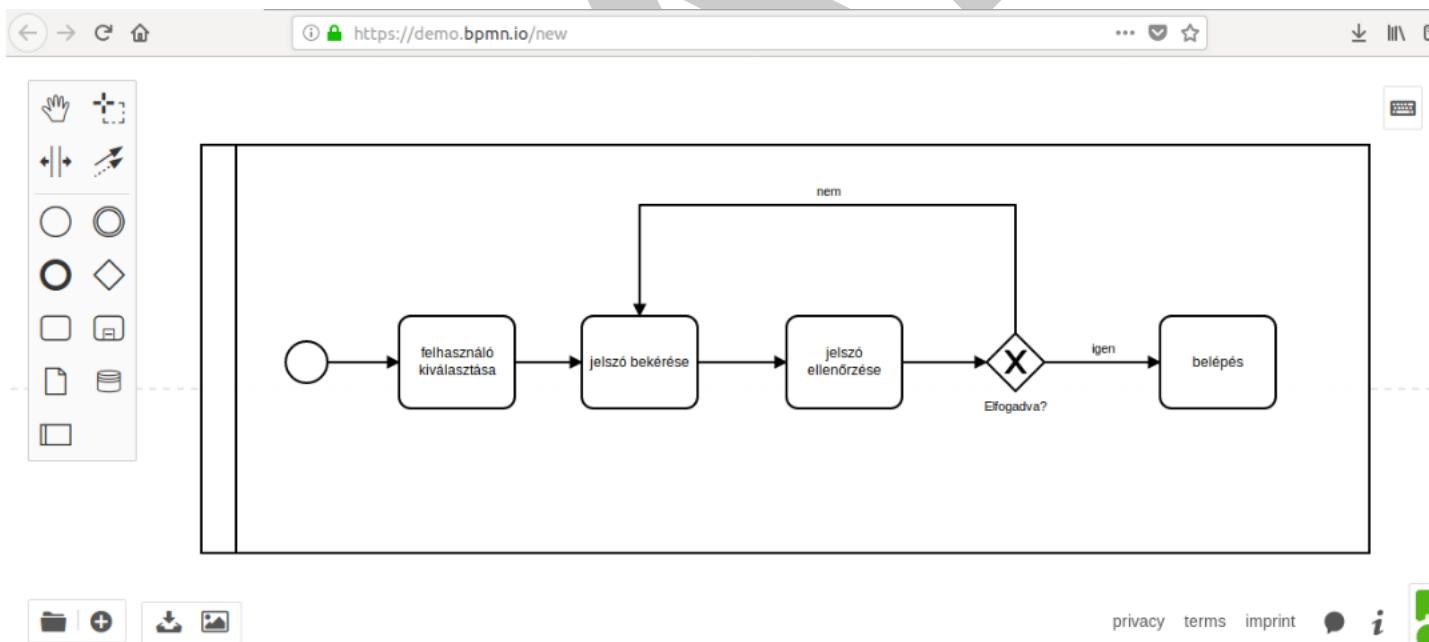
A BPMN (Business Process Model and Notation) üzleti folyamatokat reprezentál grafikusan egy üzleti folyamat modellben, amit a BPMI (Business Process Management Initiative) fejlesztett ki.

BPMN szerkesztéshez használjuk a következő online szerkesztőt: <https://demo.bpmn.io/new>

Gyakorlás képen szerkesszük meg a példában bemutatott BPMN modellt:



Ez után a feladatot teljesítve készítsünk saját modellt. (A következő BPMN modell egy operációs rendszer bejelentkezést próbál ábrázolni.)



## 6. fejezet

# Helló, Chomsky!

### 6.1. Encoding

Feladat: Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javitanitok-javat/adatok.html>

A linken található MandelbrotHalmazNagyító fordításához az ékezetes betűk használata miatt fordításnál -encoding opciót kell választanunk, hogy a fordító fel tudja dolgozni az ékezetes karaktereket is, éspedig olyat amiben megtalálhatóak az ékezetes karakterek.

A feladatnak eleget téve próbálunk ki egy opciót, amiben nem működnek az ékezetes karakterek( EUC-JP ):

The screenshot shows an IDE interface with two tabs open: `MandelbrotHalmaz.java` and `MandelbrotHalmazNagyító.java`. The `MandelbrotHalmazNagyító.java` tab contains the following code:

```

/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /**
     * A nagyítandó kijelölt területet bal felső sarka.
     */
    private int x, y;
    /**
     * A nagyítandó kijelölt terület szélessége és magassága.
     */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nyígtani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     */
    @param a [a,b]x[c,d] tartomány a koordinátája.
    @param b [a,b]x[c,d] tartomány b koordinátája.
    @param c [a,b]x[c,d] tartomány c koordinátája.
    @param d [a,b]x[c,d] tartomány d koordinátája.
*/
}

```

The terminal window below shows the compilation command and errors:

```
(base) david@david-ThinkPad-E570:~/MandelbrotHalmaz/src$ javac -encoding EUC-JP MandelbrotHalmazNagyító.java
MandelbrotHalmazNagyító.java:15: error: unmappable character for encoding EUC-JP
    /** A nagyítandó kijelölt területet bal felső sarka. */
                                         ^
MandelbrotHalmazNagyító.java:33: error: unmappable character for encoding EUC-JP
    // Az osztály konstruktornak hossza
             ^
MandelbrotHalmazNagyító.java:39: error: unmappable character for encoding EUC-JP
    // bal felső sarka:
             ^
MandelbrotHalmazNagyító.java:41: error: unmappable character for encoding EUC-JP
    // A nagyítandó kijelölt területet bal felső sarka:
```

Ebben az esetben „unmappable character” hibára hivatkozva nem fordul a program.

Most pedig próbáljuk ki UTF-8 -as encoding opciót.

```
david@david-ThinkPad-E570:~/MandelbrotHalmaz/src$ javac -encoding UTF-8 MandelbrotHalmazNagyító.java
david@david-ThinkPad-E570:~/MandelbrotHalmaz/src$
```

Igy pedig lefordul hiba nélkül.

## 6.2. I334d1c4

Feladat: Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet>

A Leet (néha l33t vagy 1337 formában is leírva), más néven eleet vagy leetspeak az angol nyelvnek egy másik ábécéje amit álltalában az interneten használnak. Az ASCII karakterek különféle kombinációját

használja a latin betűk helyett. Például a leet szó leírható 133t vagy 1337-ként, vagy pedig az eleet leírható 3133t vagy 31337-ként. A Leetet főként az angol nyelvben használják, de más nyelveken is használható pl Francia, Spanyol vagy Német.

Írunk olyan java programot ami egy szöveg betűit kicseréli a szótárban szerezőlő betűkre.

```
public class 133t{
    public static char[][] szotar= new char[][] {
        {'A','a','4'},
        {'B','b','8'},
        {'C','c','('},
        {'D','d','|','='},
        {'E','e','3'},
        {'F','f','|','='},
        {'G','g','6'},
        {'H','h','|','-','|'},
        {'I','i','1'},
        {'J','j','_','|'},
        {'K','k','|','<'},
        {'L','l','|','_'},
        {'M','m','4','4'},
        {'N','n','/','|','/'},
        {'O','o','0'},
        {'P','p','|','o'},
        {'Q','q','o'},
        {'R','r','|','2'},
        {'S','s','5'},
        {'T','t','7'},
        {'U','u','|','_','|'},
        {'V','v','|','/'},
        {'W','w','|','/','|','/'},
        {'X','x','>','<'},
        {'Y','y','|','/'},
        {'Z','z','2'}
    };

    public static String fordito(String arg){

        char[] szoveg = new char[arg.length()];

        for(int i=0; i < arg.length(); i++){
            szoveg[i]=arg.charAt(i);
        }

        StringBuffer fordított = new StringBuffer();

        for(int i=0; i<szoveg.length; i++){
            for(int j=0; j<26; j++){
                int k = 0;
                while(k < szotar[j].length) {

```

```
        if ((szoveg[i] == szotar[j][0]) || (szoveg[i]== ↔
            szotar[j][1])){
                for(int l=0; l<szotar[j].length-2; l++) {
                    fordított.append(szotar[j][l+2]);
                }
                break;
            }
            k++;
        }
    }

    return fordított.toString();
}

public static void main(String[] args){

    if(args.length != 1) {
        System.out.println("l33t.java <szöveg>");
        System.exit(-1);
    }

    String fordított = fordito(args[0]);

    System.out.println(fordított);
}
}
```



The screenshot shows an IDE interface with a Java file named `l33t.java` open. The code defines a class `l33t` with a static method `fordito` that takes a string argument and returns a decoded leet string. The main part of the code is a large switch statement mapping characters from the standard alphabet to their leet equivalents. Below the code, a terminal window shows the compilation of the Java file with `javac l33t.java` and the execution of the program with `java l33t`, passing the string "batfaimester" as an argument. The output shows the decoded leet string "847|=41443573|2".

```
public class l33t{
    public static char[][] szotar= new char[][] {
        {'A','a','4'},
        {'B','b','8'},
        {'C','c','('},
        {'D','d','|_|')'},
        {'E','e','3'},
        {'F','f','|_-'},
        {'G','g','6'},
        {'H','h','|_/_|_/'},
        {'I','i','1'},
        {'J','j','_|_/'},
        {'K','k','|_<'},
        {'L','l','|_/_'},
        {'M','m','4444'},
        {'N','n','/_/_|_/'},
        {'O','o','0'},
        {'P','p','|_|'0'},
        {'Q','q','0'},
        {'R','r','|_|'2'},
        {'S','s','5'},
        {'T','t','7'},
        {'U','u','|_|/_|_/'},
        {'V','v','|_|/_/'},
        {'W','w','|_|/_/_|_/'},
        {'X','x','>_<'},
        {'Y','y','^_/_/'},
        {'Z','z','2'}
    };
    public static String fordito(String arg){
        l33t > fordito()
    }
}

Terminal: Local +  

(base) david@david-ThinkPad-E570:~/l33t/src$ javac l33t.java  

(base) david@david-ThinkPad-E570:~/l33t/src$ java l33t leet  

|_337  

(base) david@david-ThinkPad-E570:~/l33t/src$ java l33t batfaimester  

847|=41443573|2  

(base) david@david-ThinkPad-E570:~/l33t/src$
```

A program bekér argumentumként egy szöveget, amit leet nyelvre fordít át.

### 6.3. Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: [https://www.tankonyvtar.hu/en/tartalom/tkt/javatitok-javat/ch03.html#labirintus\\_jatek](https://www.tankonyvtar.hu/en/tartalom/tkt/javatitok-javat/ch03.html#labirintus_jatek)

A feladat tehát az, hogy készítsünk el egy teljes képernyős Java programot, ami egy teljes képernyős üres képernyő is lehet, ennek megfelelően írunk olyan teljes képernyős Java programot, ami elfoglalja a teljes képernyőt.

```
import java.awt.Canvas;
import java.awt.Dimension;
```

```
import java.awt.Frame;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class FullScreen {

    private static Frame frame;
    private static Canvas canvas;

    private static int canvasWidth = 0;
    private static int canvasHeight = 0;

    private static void makeFullscreen() {
        GraphicsEnvironment env = GraphicsEnvironment.←
            getLocalGraphicsEnvironment();
        GraphicsDevice gd = env.getDefaultScreenDevice();

        if(gd.isFullScreenSupported()) {
            gd.setFullScreenWindow(frame);
        }
    }

    public static void init (){
        frame = new Frame();
        canvas = new Canvas();
        Dimension dim = new Dimension(canvasWidth, ←
            canvasHeight);

        canvas.setPreferredSize(dim);
        frame.add(canvas);
        //fejléc eltüntetése
        frame.setUndecorated(true);
        //ha lenne content
        frame.pack();
        //méretezés eltüntetése
        frame.setResizable(false);
        //képernyő közepére helyezés
        frame.setLocationRelativeTo(null);

        makeFullscreen();

        //ha lenne content
        frame.setVisible(true);

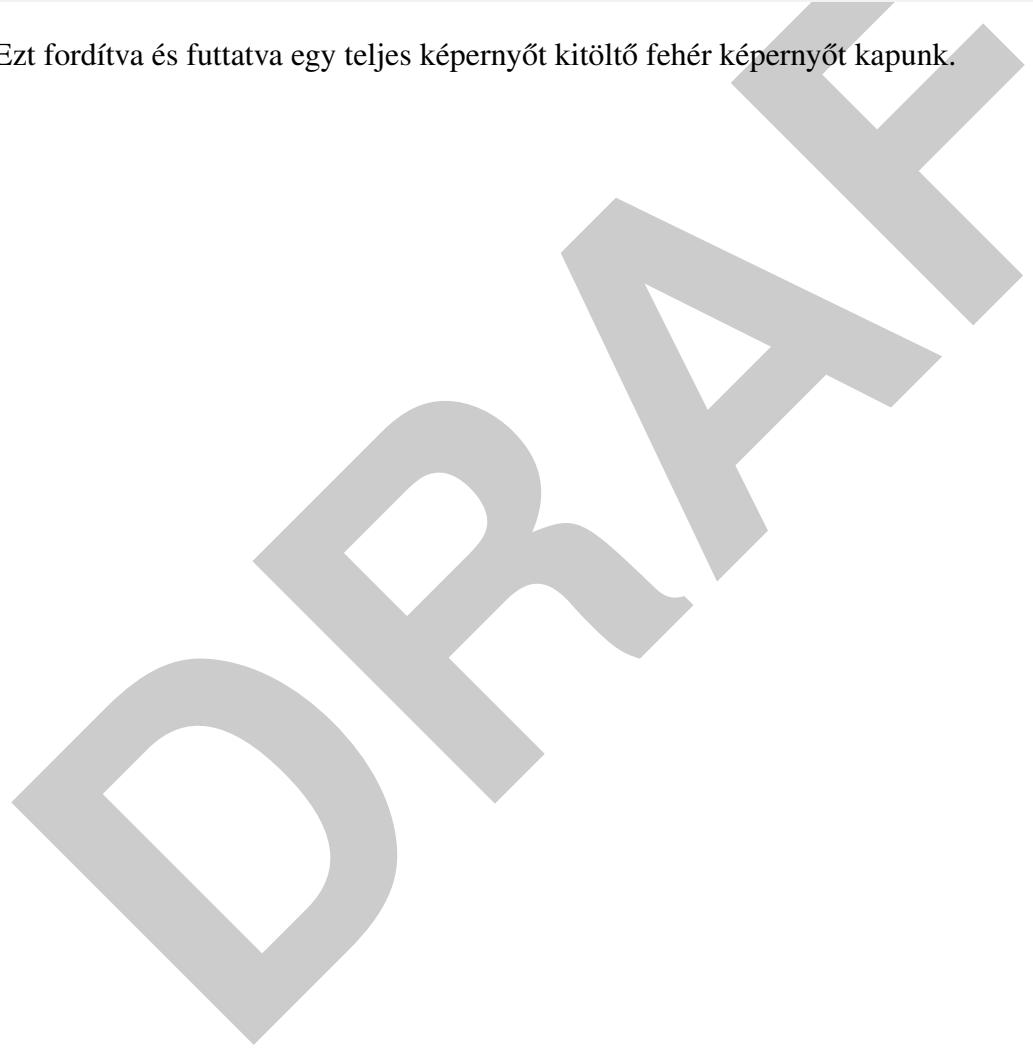
        //ablak bezáró gomb
        frame.addWindowListener(new WindowAdapter () {
            public void windowClosing (WindowEvent e) {
```

```
        quit();
    }
} );

public static void main(String[] args) {
    init();
}

public static void quit() {
    System.exit(0);
}
}
```

Ezt fordítva és futtatva egy teljes képernyőt kitöltő fehér képernyőt kapunk.



## 7. fejezet

# Helló, Stroustrup!

### 7.1. JDK osztályok

Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

A JDK (Java Development Kit) a Sun Microsystems terméke, amit a Java fejlesztőknek szántak. A JDK az egyik legnépszerűbb fejlesztőeszköz.

A JDK rengeteg programozási eszközökt tartalmaz, például a java fordítóprogramot is, a javac-t.

A feladat megoldásához írunk olyan Boost c++ programot, ami megszámolja és kilistázza a JDK osztályokat, majd az Open JDK 10.0.2-es verziójában szereplő src.zip-et csomagoljuk ki és engedjük rá a programot.

forras1:

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

using namespace boost::filesystem;

std::vector<std::string> listOfFiles;
int count = 0;

//std::vector<std::string> getAllFilesInDir(std::string dirPath) {
void countFiles(std::string dirPath){
    path p(dirPath);

    try {
        if (exists(p) && is_directory(p)) {
            for (directory_entry& x : directory_iterator(p)) {
                //listOfFiles = getAllFilesInDir(x.path().string());
                countFiles(x.path().string());
            }
        }
    }
}
```

```
        }
    }
else {
    //listOfFiles.push_back(p.string());
    count++;
}
}

catch (std::system_error & e) {
    std::cerr << "Exception :: " << e.what();
}
//return listOfFiles;
}

int main(int argc, char* argv[]){
    std::string dirPath = argv[1];

    /*std::vector<std::string> listOfFiles = getAllFilesInDir(dirPath);

    int k = 0;

    for (auto str : listOfFiles) {
        std::cout << str << std::endl;
    }

    std::cout << listOfFiles.size();*/
    countFiles(dirPath);
    std::cout << count ;
}
```



The screenshot shows an IDE interface with two windows. The top window is titled 'PolarGenerator.java' and contains the following Java code:

```
1  public class PolarGenerator {
2      boolean nincsTarolt = true;
3      double tarolt;
4      public PolarGenerator() {
5          nincsTarolt = true;
6      }
7      public double kovetkezo() {
8          if(nincsTarolt) {
9              double u1, u2, v1, v2, w;
10             do {
11                 u1 = Math.random();
12                 u2 = Math.random();
13                 v1 = 2 * u1 - 1;
14                 v2 = 2 * u2 - 1;
15                 w = v1 * v1 + v2 * v2;
16             } while(w > 1);
17             double r = Math.sqrt(-2 * Math.log(w) / w);
18             tarolt = r * v2;
19             nincsTarolt = !nincsTarolt;
20             return r * v1;
21         } else {
22             nincsTarolt = !nincsTarolt;
23             return tarolt;
24         }
25     }
26 }
```

The bottom window is titled 'PolarGenerator' and shows the output of the program:

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
-2.0424536265181845
0.05296707421087484
0.796903445383798
0.15118134693433039
-0.6250452487773279
-0.4346800843293371
-0.5787399242026225
1.0172412827190924
1.5417401377874569
0.5052621983341685

Process finished with exit code 0
```

## 7.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékkadásra!

**Konstruktor:** Az objektum működéséhez szükséges erőforrásokat foglalja le.

**Destruktor:** Erőforrásokat szabadít fel, az objektum megsemmisülése előtt.

**Másoló konstruktor:** Az objektum működéséhez szükséges erőforrásokat foglalja le, majd az objektumot egy másik példány másolataként létrehozza.

A feladat megoldásához ítjunk olyan c++ programot amiben a mozgató konstruktort a mozgató értékkadásra alapozzuk.

```
#include<iostream>
using namespace std;

class Point
{
private:
    int x;
public:
    //Constructor
    Point(int x1) {
        x = x1;
    }

    // Copy constructor
    Point(const Point& p2) {
        cout << "Copy constructor called" << " \n";
        x = p2.x;
    }

    //Move constructor
    Point(const Point&& p2) {
        cout << "Move constructor called" << " \n";
        *this = move(p2.x);
    }

    //Move assign
    Point& operator= (Point&& p2) {
        cout << "Move assignment called" << " \n";
        x = p2.x;
    }

    int getX() {
        return x;
    }
};

int main()
{
    //Ctor
    Point p1(10);
    //Move ctor
    Point p2 = move(p1);
    //Copy ctor
    Point p3 = p1;

    cout << "p1.x = " << p1.getX() << " \n";
    cout << "p2.x = " << p2.getX() << " \n";
    cout << "p3.x = " << p3.getX() << " \n";
}
```

```
    return 0;  
}
```

### 7.3. Összefoglaló: JDK osztályok

A Feladat az volt, hogy írunk olyan Boost C++ programot amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve). A JDK (Java Development Kit) a Sun Microsystems terméke, amit a Java fejlesztőknek szántak. A JDK az egyik legnepszerűbb fejlesztőeszköz. A JDK rengeteg programozási eszközöt tartalmaz, például a java fordítóprogramot is, a javac-t.

Mivel előző félévben alap követelmény volt a „fénykard” névre hallgató boostot használó program megírása, ezért nem volt túl idegen ez a feladat. Csak annyi a feladat, hogy egy mappán belül szerepő .java kiterjesztésű fájlokat számoljuk össze.

A megírt program rekurziót használ a mappa bejárására, a következő képpen: a mappában az összes fájlt és mappát megvizsgálja, ha talál egy .java kiterjesztésű fájt, növel egy változó értéket, ha pedig mappát talál abba belép és ugyan ezt végrehajtja addig, amíg elfogynak a mappák. A feladathoz az Open JDK 10.0.2-es verzióját vizsgáljuk meg. Kicsomaguljuk az src.zip-et és ráengedjük a megírt programot. A program kiírja a talált .java kiterjesztésű fájlok számát, amit 22480, ez azt jelenti, hogy az Open JDK 10.0.2-es verziója 22480 darab osztályt tartalmaz. Mivel a feladat megoldásának nem szempontja, hogy tároljuk és kiírjuk az összes .java kiterjesztésű fájlt, ezért ezt nem tesszük meg és csak megszámoljuk azokat, így a program gyorsan lefut.

## 8. fejezet

# Helló, Gödel!

### 8.1. STL map érték szerinti rendezése

Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

A tárolók az STL (Standard Template Library)-nek fontos részét képezik. Az STL tárolók a beépített tömbökkel és a kézzel készített láncolt adatszerkezetekkel szemben olyan adatszerkezetek, amelyek különféle tárolási megoldással biztonságosan és hatékonyan tárolják az adatokat.

A map egy variánsa a multimap, amelyben egy kulcs többször is szerepelhet, így megjelenik a kulcs számoság.

Mivel az STL map alapból kulcsérték szerint rendezzi önmagát, ezért az érték szerinti rendezést egyszerűen úgy oldjuk meg, hogy átrakjuk a map elemeit egy pair vektorba, aztán magáta vektort rendezzük forrás:

```
#include <iostream>
#include <map>
#include <vector>

using namespace std;

int main() {

    vector<pair<int, int>> sorted;
    int temp1 = 0;
    int temp2 = 0;
    std::map<int, int> a;
    a[1] = 2;
    a[45] = 10;
    a[5] = 9;
    a[54] = 3;
    a[3] = 15;
    a[15] = 100;
    a[8] = 5;
```

```
a[22] = 13;

//map kiiratása
for (const auto &p : a) {
    std::cout << "a[" << p.first << "] = " << p.second << '\n';
}
std::cout<<endl<<"érték szerint rendezve:"<<endl;

//map berakása vektorba
for (const auto &p : a) {
    sorted.push_back(pair<int, int> (p.first, p.second));
}

//vektor rendezése
for(int i = 0; i < sorted.size(); i++) {
    for(int j = 0; j <sorted.size(); j++) {
        if(sorted[i].second < sorted[j].second) {
            temp1 = sorted[i].first;
            temp2 = sorted[i].second;
            sorted[i].first = sorted[j].first;
            sorted[i].second = sorted[j].second;
            sorted[j].first = temp1;
            sorted[j].second = temp2;
        }
    }
}

//rendezett vektor kiiratása
for (const auto &p : sorted) {
    std::cout << "a[" << p.first << "] = " << p.second << '\n';
}

return 0;
}
```

## 8.2. Alternatív Tabella rendezése

Mutassuk be a [https://progpter.blog.hu/2011/03/11/alternativ\\_tabella](https://progpter.blog.hu/2011/03/11/alternativ_tabella) a programban a java.lang Interface Comparable

<T>

szerepét!

A Csatlakozó objektumban két értéket tárolunk, ezt a két értéket hasonlíthatjuk össze a compareTo() függvényel. Az „ertek” változó kereül összehasonlításra a többi objektumhoz tartozó „ertek” változóval, így háromféle eredmény születhet és ennek megfelelően a következő kimenetek alakulhatnak ki: ha az adott „ertek” változó értéke kisebb mint a többié, akkor a kimenet 1, ha nagyobb, akkor a kimenet -1 illetve, ha egyenlő akkor a kimenet 0.

Fordítsuk és futtassuk a Wiki2Matrix programot:

Majd a kapott értékmátrixot másoljuk be az AlternativTabella programba, és így forditsuk, futtassuk a programot:

The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure with files like `AlternativTabella.java`, `.idea`, `out`, `src`, and `AlternativTabella.iml`.
- Code Editor:** Displays the `AlternativTabella.java` file containing a main method that initializes a double matrix `L` with specific values.
- Run Output:** Shows the terminal output of the application running. It includes:
  - Java command: `/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...`
  - Iteration loop output:
    - norma = 0.05918759643574294
    - φsszeg = 1.0
    - norma = 0.014871507967965858
    - φsszeg = 0.999999999999999
    - norma = 0.006686056768932613
    - φsszeg = 1.0
    - norma = 0.007776749198562133
    - φsszeg = 1.0
    - norma = 0.007661483555477314
    - φsszeg = 0.999999999999999
    - norma = 0.007501657116770109
    - φsszeg = 0.999999999999999

## 8.3. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

A lambda kifejezések segítségével név nélküli függvényeket tudunk megadni, ezek rövid egyszer felhasznát függvények. Egy lambda kifejezések általánosan:

```
[/*capture*/]  /*parameters*/
{
    /*body*/
}
```

Mint látható visszatérítési értéket nem szükséges megadni, ezt majd a fordító kikövetkezteti. Az OOCWC feladatban említett kód részlet:

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

A lambda kifejezést egy rendezés során tünik fel, a rendezést az std::sort segítségével valósítjuk meg. Ez a kódcsipet esetén a gangsters vektor kerül rendezésre. Paraméterként két Gangster objektumot kap, visszatérítési érték igaz, ha az x közelebb van a rendőrhöz, mint az y. Így miután lefut a veltorban a legelső helyn az a gengszter lesz aki legközelebb van a rendőrhöz.

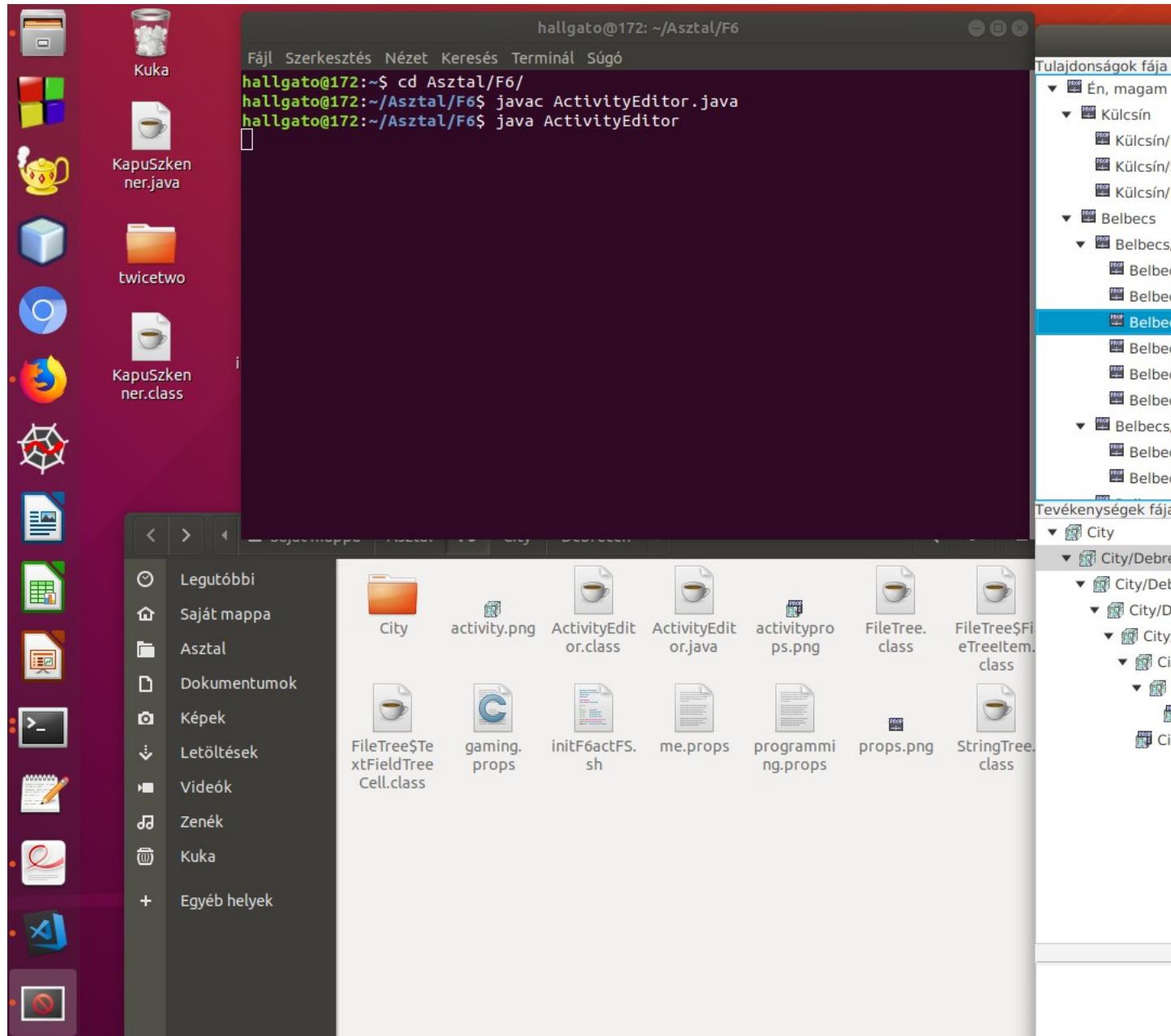
## 9. fejezet

### Helló,!

#### 9.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6> <https://www.twitch.tv/videos/222879467>

A FUTURE tevéknység editor a hallgatók aktivitásainak illetve tulajdonságainak nyomon követésére szolgál átlátható fa szerkezetben ábrázolva. A feladat az, hogy javítsunk valamit az ActivityEditor.java programon. Mivel javításnak számít újabb tulajdonságok felvétele ezért tegyük ezt.



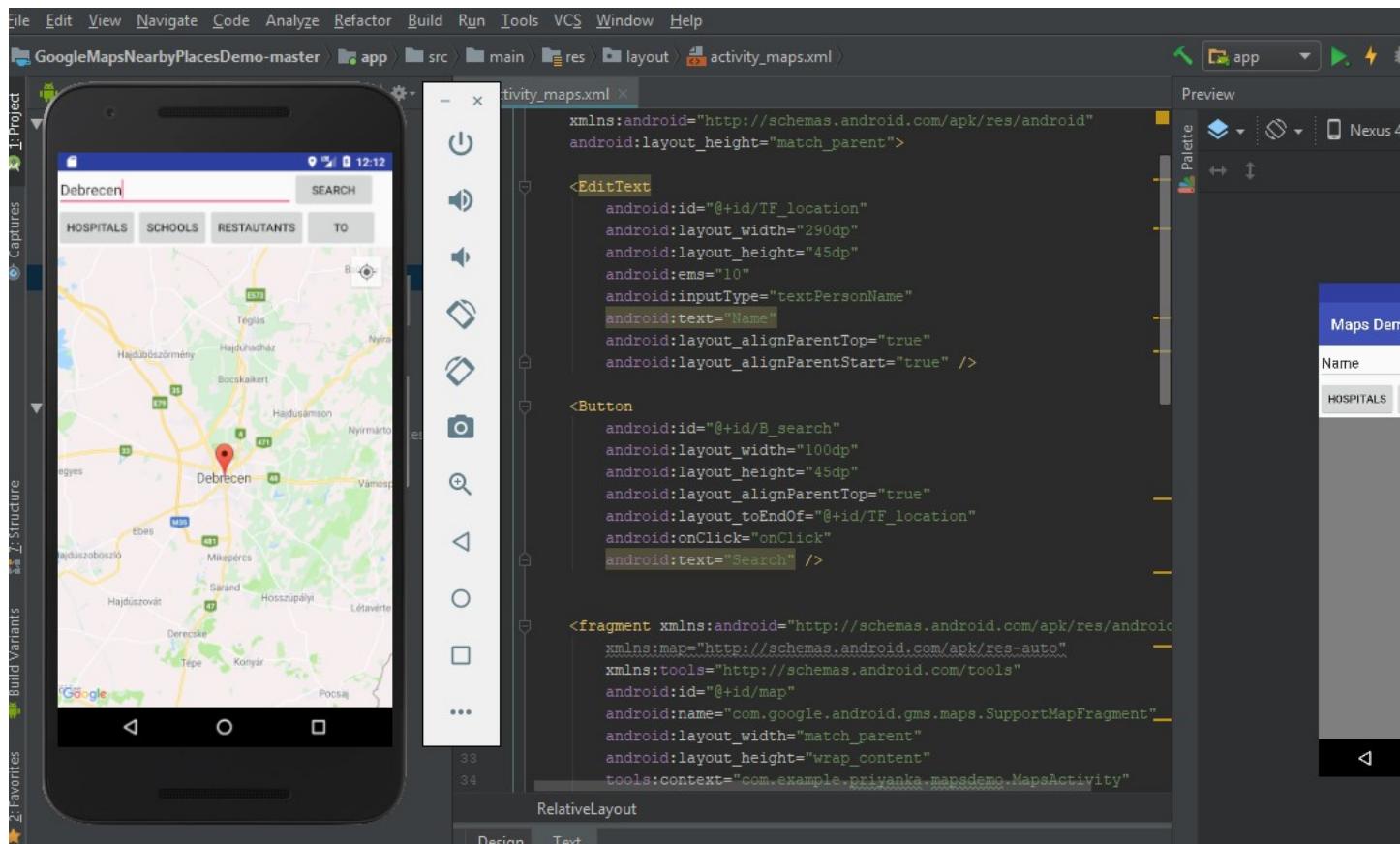
## 9.2. OSM térképre rajzolása

Feladat: Debrecen térképre dobjunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537> (de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/nbatfai/robocar-emulator/tree/master/justine/rcwin> is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Az OSM (Open Street Map) térkép célja egy szabadon felhasználható és szerkeszthető térkép létrehozása. A feladat teljesítéséhez nézzünk utána hogyan működnek az ilyes fajta programok és hozzunk létre egyet.

Android map készítés a következő youtube tutorial alapján: <http://y2u.be/Ot8D-GZ8qfY>

A végeredmény:



### 9.3. OOCWC Boost ASIO hálózatkezelése

Feladat: Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

```

std::vector<Gangster> gangsters;

while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n ) == 4 )
{
    nn += n;
    gangsters.push_back ( Gangster {idd, f, t, s} );
}

```

A c könyvtár függvénye:

```
int sscanf(const char *str, const char *format, ...)
```

formázott string imputot olvas be. Az adott példában a sscanf 4 paramétert vár beolvasásra (a %n visszatérési értéke az eddig beolvasott karakterek száma). Akkor hozunk csak létre új Gangster objektumot ha sikerült mind a 4 paraméter beolvasása. Az nn változóhoz minden sikeres objektum létrehozás esetén hozzáadjuk az n változó értékét. Tehát az nn változóban az összes beolvasott karakterek számát tároljuk. Erre azért van szükség, hogy a sscanf az imput string azon részét olvassa ami még nem került feldolgozásra.

# 10. fejezet

## Helló, Lauda!

### 10.1. Port scan

Feladat: Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu-hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id52728>

A port szkennelő forrás az argumentumként megadott gép portjain TCP kapcsolatot próbál meg kialakítani, ha ez nem sikerül akkor, egy IOException kivétel jön létre. A kivételkezelés szerepe tehát eldönteni, hogy figyeli-e az adott porotot egy folyamat.

Futtassuk ezt le, argumentumként saját gépet megadva, aztán a grep segítségével nézzük meg, hogy milyen portokat figyel.

Forras:

```
public class KapuSzkenner {
    public static void main(String[] args) {
        for(int i=0; i<1024; ++i)
            try {
                java.net.Socket socket = new java.net.←
                    Socket(args[0], i);

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

                socket.close();

            } catch (Exception e) {
                System.out.println(i + " nem figyeli");
            }
    }
}
```

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Kapuszkennerr". It contains a "src" directory which includes "Kapuszkennerr.java", ".idea", "out", and "Scratches and Consoles". There is also an "External Libraries" section.
- Kapuszkennerr.java Content:**

```
public class Kapuszkennerr {
    public static void main(String[] args) {
        for(int i=0; i<1024; ++i)
            try {
                java.net.Socket socket = new java.net.Socket(args[0], i);
                System.out.println(i + " figyeli");
                socket.close();
            } catch (Exception e) {
                System.out.println(i + " nem figyeli");
            }
    }
}
```
- Run Tab:** The "Run" tab is selected, showing the output of the application. The output window displays:

```
1005 nem figyeli
1006 nem figyeli
1007 nem figyeli
1008 nem figyeli
1009 nem figyeli
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
```

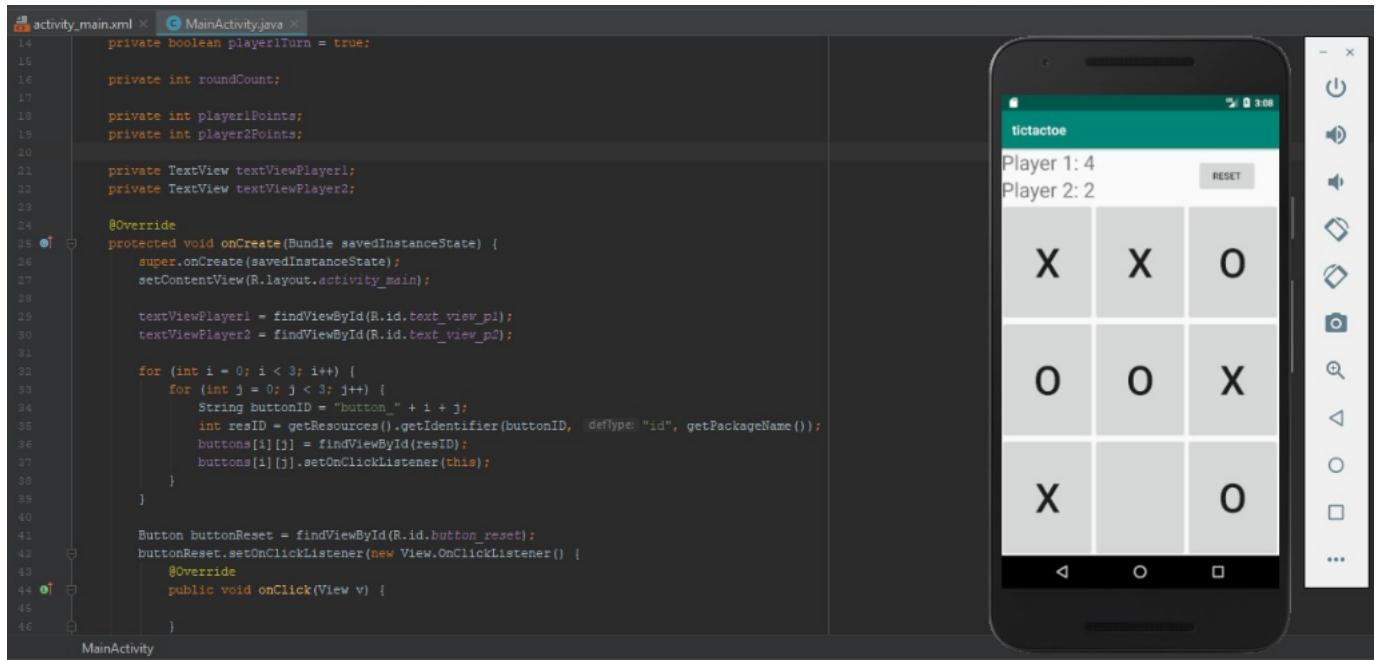
Process finished with exit code 0

## 10.2. Android Játék

Írunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Hello, Android!” feladatára!

A feladat szerint tehát egy működő androidos játékot kell írni, én ezt a következő youtube tutorial alapján tettem: <https://www.youtube.com/watch?v=apDL78MFR3o>

A játék egy tictactoe játék, ami a játék szabályai alapján működik, számolja a játékosok pontjait, valami egy reset funkció is bele van építve, ami az eddig összegyűjtött pontokat nullázza.



### 10.3. Junit teszt

A [https://progpter.blog.hu/2011/03/05/labormeres\\_otton\\_hogyan\\_dolgozok\\_fel\\_egen\\_pedat](https://progpter.blog.hu/2011/03/05/labormeres_otton_hogyan_dolgozok_fel_egen_pedat) poszt kezze számított mélységet és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

A JUnit egy egységtesztelő keretrendszer. A teszt során lehetőség van hogy a programunkat ellenőrizzük, hogy bizonyos esetekben minden rendben működik benne. Ezzel egyes Unitokat, azaz részeket szoktunk tesztelni, egy osztályokat vagy függvényeket. A feladatunk hogy az LZWBInFa.java programbeli mélységet és a szórást kiszámító függvényekhez írunk egy JUnit tesztet. Így egyszerüen le tudunk ellenorozni majd bármilyen függvényt ami az LZWBInFa programban van. A programban a Progpter-en lévő értékeket használtam.

## 11. fejezet

# Helló, Calvin!

### 11.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, [https://progpater.blog.hu/2016/11/13/hello\\_sbol](https://progpater.blog.hu/2016/11/13/hello_sbol) Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

A TensorFlow a gépi tanulás megvalósítása. Széles körű algoritmusok megvalósítására képes, például beszédfelismerés, számítógépes látás, robotikában, de akár az agykutatásban is alkalmazható és a számítástudomány más területein is. A feladat szerint tehát telepítünk a tensorflow-t aztán futtassuk a python programot és érjük el, hogy álltalunk rajzolt számjegyet is ismerjen fel:

```
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %

-----
-- A halozat tesztelese
-- Pontosság: 0.9194

-----
-- A MNIST 42. tesztkepene felismerese, mutatom a szamot, a tovabbepeshez csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 4

-----
-- A sajat kezi 8-asom felismerese, mutatom a szamot, a tovabbepeshez csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 5
```

A feladatban bemutatott program fekete háttérű számjegyeket ismer fel, ehhez a TensorFlow-ot használja. A Google Brain csapat által megvalósított TensorFlow a numerikus számításhoz és a nagy gépi tanuláshoz létrehozott nyílt forrású könyvtár. A TensorFlow segítségével a számítások eloszthatóak a CPU-kon, a GPU-kon vagy különböző eszközökön. A TensorFlow képes a mély neurális hálózatok kiépítésére és működtetésére.

A TensorFlow a gépi tanulás megvalósítása. Széles körű algoritmusok megvalósítására képes, például beszédfelismerés, számítógépes látás, robotikában, de akár az agykutatásban is alkalmazható és a számítástudomány más területein is. A gépi tanulás fejlesztésére a hardver teljesítmények folyamatos növekedése pozitívan hat.

A gépi tanulás megértéséhez először nézzük meg, hogyan tanulnak az emberek. Az agyadban található neuronok az idegrendszer legfontosabb feldolgozó egységei. Egymással állnak kapcsolatban bonyolult

hálózatot alkotva. Az emberi tanulás során az agyban található neuronok között kapcsolatok más néven szinapszisok jönnek létre vagy szűnnek meg. A gépek kis túlzással ugyanígy tanulnak. A gépi tanuló algoritmusok is neuronokból épül fel amelyeket összekötünk egymással, majd a gépi tanulás során ezeket az összeköttetéseket módosítjuk. Ilyen esetben az a cél, hogy úgy alakítsuk ki az összeköttetéseket, hogy az adott feladatot minél jobban hajtsa végre.

## 11.2. Android telefonra a TF objektum detektálója

Feladat: T elepítsük fel, próbáljuk ki!

A feladat szerint tehát importáljuk AndroidStudioba a tensorflowos appot és érjük el, hogy működjön is. Több alprogram elérhető: TF Classify, TF Detect, TF Speech, TF Stylize. A Feladat teljesítéséhez próbáljuk ki a TF Classify programot, ami megpróbálja beazonosítani a tárgyakat.



11.3.

11.4.

### III. rész

#### Irodalomjegyzék

DRAFT

## 11.5. Általános

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

## 11.6. C

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

## 11.7. C++

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

## 11.8. 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 UDPORG 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 érdeklődésekért és hasznos észrevételeikért.

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