

Jegyzőkönyv

Szilágyi-Czumbil Ede Balázs

Jegyzőkönyv

Szilágyi-Czumbil Ede Balázs

Table of Contents

I.	1
1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroway!”	5
OO szemlélet	5
„Gagyí”:	6
Yoda	8
4. 2. hét - „Helló, Liskov!”	11
Liskov helyettesítés sértése	11
Szülő-gyerek	12
Hello, Android!	15
5. 3. hét - „Helló, Mandelbrot!”	19
Reverse engineering UML osztálydiagram	19
Forward engineering UML osztálydiagram	20
BPMN	20
6. 4. hét - „Helló, Chomsky!”	22
Encoding	22
l33t	23
Fullscreen	25
7. 5. hét - „Helló, Stroustrup!”	26
JDK osztályok	26
Változó argumentumszámú ctor	27
Összefoglaló	28
8. 6. hét - „Helló, Gödel!”	31
STL map érték szerinti rendezése	31
Alternatív Tabella rendezése	31
Prolog családfa	33
9. 7. hét - „Helló, !”	35
FUTURE tevékenység editor	35
OOCWC Boost ASIO hálózatzkezelése	36
BrainB	36
10. 8. hét - „Helló, Lauda!”	38
Port scan	38
AOP	39
Android Játék	40
11. 9. hét - „Helló, Calvin!”	42
MNIST	42
CIFAR-10	43
Android telefonra a TF objektum detektálója	44

List of Tables

7.1. Copy ctor vs Move ctor	30
-----------------------------------	----

Part I.

Table of Contents

1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroy!	5
OO szemlélet	5
„Gagy!	6
Yoda	8
4. 2. hét - „Helló, Liskov!	11
Liskov helyettesítés sértése	11
Szülő-gyerek	12
Hello, Android!	15
5. 3. hét - „Helló, Mandelbrot!	19
Reverse engineering UML osztálydiagram	19
Forward engineering UML osztálydiagram	20
BPMN	20
6. 4. hét - „Helló, Chomsky!	22
Encoding	22
l33t	23
Fullscreen	25
7. 5. hét - „Helló, Stroustrup!	26
JDK osztályok	26
Változó argumentumszámú ctor	27
Összefoglaló	28
8. 6. hét - „Helló, Gödel!	31
STL map érték szerinti rendezése	31
Alternatív Tabella rendezése	31
Prolog családja	33
9. 7. hét - „Helló, !	35
FUTURE tevékenység editor	35
OOCWC Boost ASIO hálózatkészítése	36
BrainB	36
10. 8. hét - „Helló, Lauda!	38
Port scan	38
AOP	39
Android Játék	40
11. 9. hét - „Helló, Calvin!	42
MNIST	42
CIFAR-10	43
Android telefonra a TF objektum detektálása	44

Chapter 1. Infók

Neptun kód: CMY9W3

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

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

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

Java, C++ összehasonlítás

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

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

Python

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

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

Chapter 3. 1. hét - „Helló, Arroway!”

1. hét Az objektumorientált paradigma alapfoglamai. Osztály, objektum, példányosítás.

OO szemlélet

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

A kód:

```
public class PolarGenerator {

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator() {
        nincsTarolt = true;
    }

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

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

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

            tarolt = r * v2;

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

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

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

A program kezdésként két változót állít el#. Magát a tárolt számot(double) és egy boolean típusu változót ami tárolja hogy van-e változó.Ezután ellen#rzi hogy van e tárolt változó,ha van akkor generál 2 random számot amivel elvégzi az adott m#veletet. Ezt addig folytatja amíg a kapott eredmény kisebb lesz 1nél . Ha a nincs tárolt változó false akkor visszaadja a tárolt változóban lévő# értéket.

Miután futtatuk:

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

„Gagyi”:

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

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

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

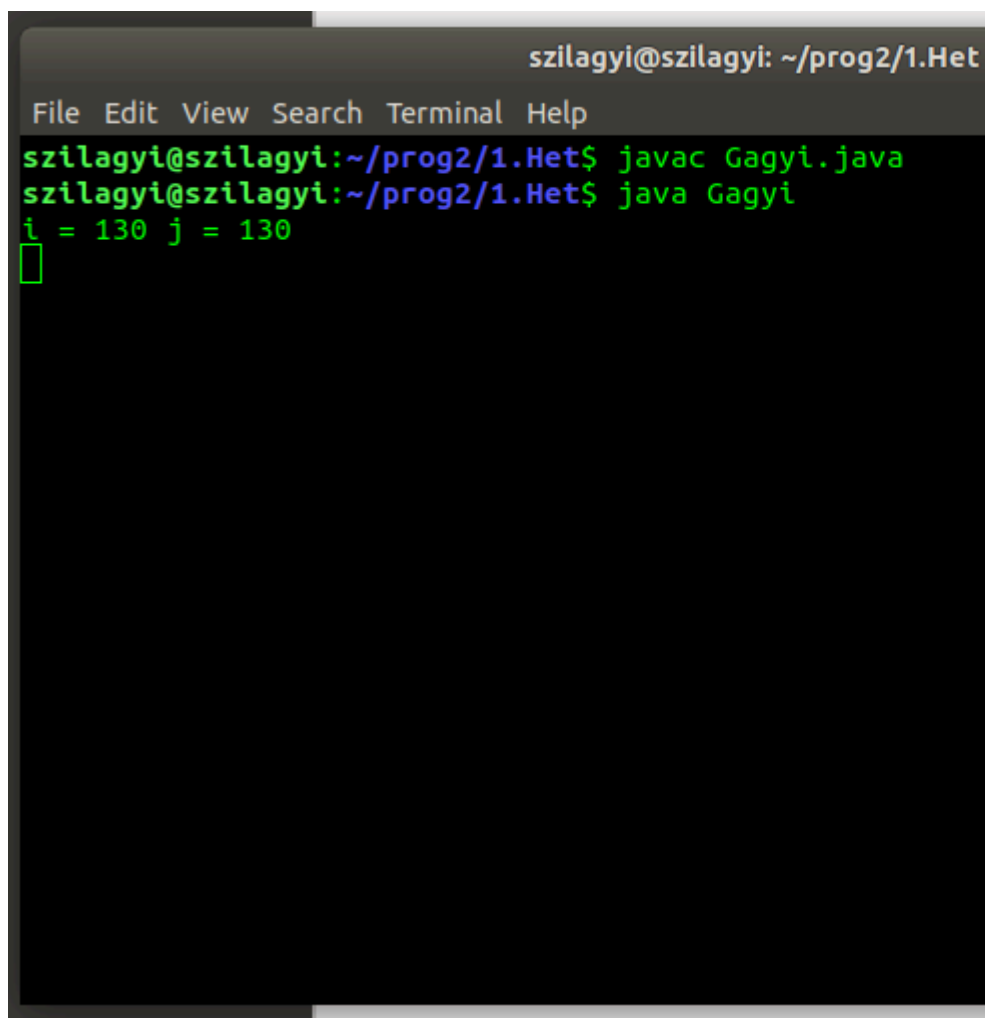
E miatt ha a programunkban a [-128,127] intervallumon kívüli értéket adunk meg akkor az egyenl#ség hamis lesz mivel két új obiekum fog létrejönni és emiatt végtelen ciklust kapunk.

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

a program ami végtelen ciklust ad:

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

lefuttatva:



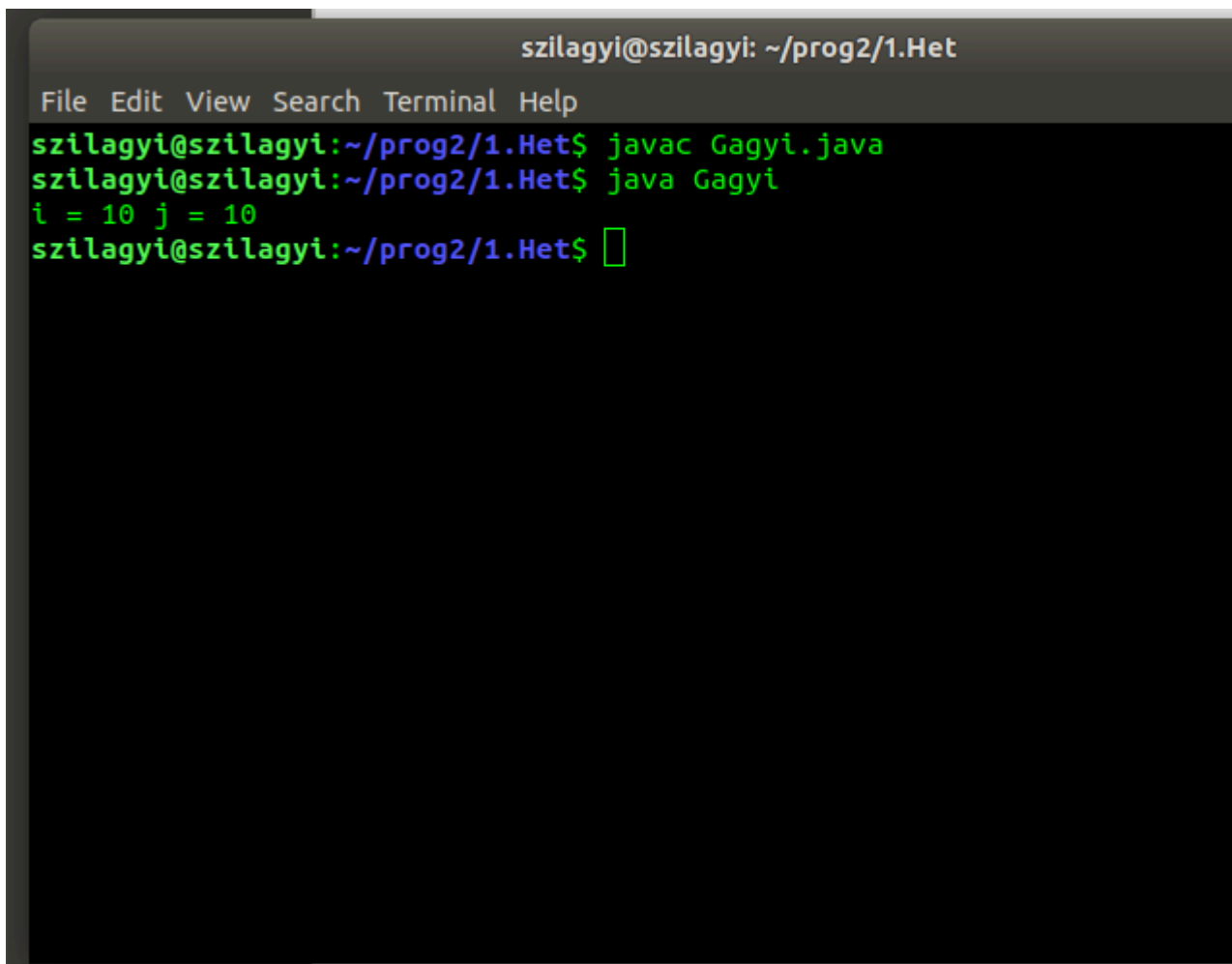
```
szilagyi@szilagyi: ~/prog2/1.Het  
File Edit View Search Terminal Help  
szilagyi@szilagyi:~/prog2/1.Het$ javac Gagyi.java  
szilagyi@szilagyi:~/prog2/1.Het$ java Gagyi  
i = 130 j = 130  
█
```

a program ami nem végtelen ciklust ad:

```
public class Gagyi {
```

```
public static void main(String[] args) {  
  
    Integer i = 10;  
    Integer j = 10;  
  
    System.out.println("i = " + i + " j = " + j);  
  
    while(i <= j && i >= j && i != j) {  
  
    }  
}  
}
```

lefuttatva:



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

Yoda

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

A kód:

```
public class Yoda {
```

```
public static void main(String[] args) {  
  
    final String str = null;  
  
    try {  
        if(str.equals("...")) {  
            //Do something  
        }  
        System.out.println("1. Success");  
  
    } catch(Exception e) {  
        System.err.println(e.getMessage());  
    }  
  
    try {  
        if("..." equals(str)) {  
            //Do something  
        }  
        System.out.println("2. Success");  
  
    } catch(Exception e) {  
        System.err.println(e.getMessage());  
    }  
}
```

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

Az el#z# programrészlet pont ezt próbálja szemléltetni. amikor el#sször próbáljuk ellen#rizni az egyenl#séget NullPointerException-el leáll mivel megsértettük a Yoda condition-t (null), de mikor megcseréltük a sorrendet már sikerrel jártunk (Success)

Lefuttatva:

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

Chapter 4. 2. hét - „Helló, Liskov!”

2. hét Örökl#dés, osztályhierarchia. Polimorfizmus, metódust#lterhelés. Hatáskörkezelés. A bezárási eszk#zrendszer, láthatósági szintek. Absztrakt osztályok és interfészek.

Liskov helyettesítés sértése

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

C++ kódcsipet amely megsérti a Liskov elvet:

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

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

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

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

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

    Sas sas;
    program.fgv ( sas );

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

Így lenne helyes:

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

```
// ez a két osztály alkotja a "P programot" az LPS-ben
class Program {
public:
    void fgv ( Madar &madar ) {
        // madar.repul(); a madár már nem tud repülni
        // s hiába lesz a leszármazott típusoknak
        // repül metódusa, azt a Madar& madar-ra úgysem lehet hívni
    }
};

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

class Sas : public RepuloMadar
{};

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

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

    Sas sas;
    program.fgv ( sas );

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

Szülo-gyerek

Írjunk Szül#-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az #sön keresztül csak az #s üzenetei küldhet#ek!

Ez a feladat csupán azt demonstrálná, hogy nem lehetséges egy adott szülő referencián keresztül, ami egy # gyerek objektumára hivatkozik, meghívni gyermeke egy olyan metódusát amit o maga nem definiált.

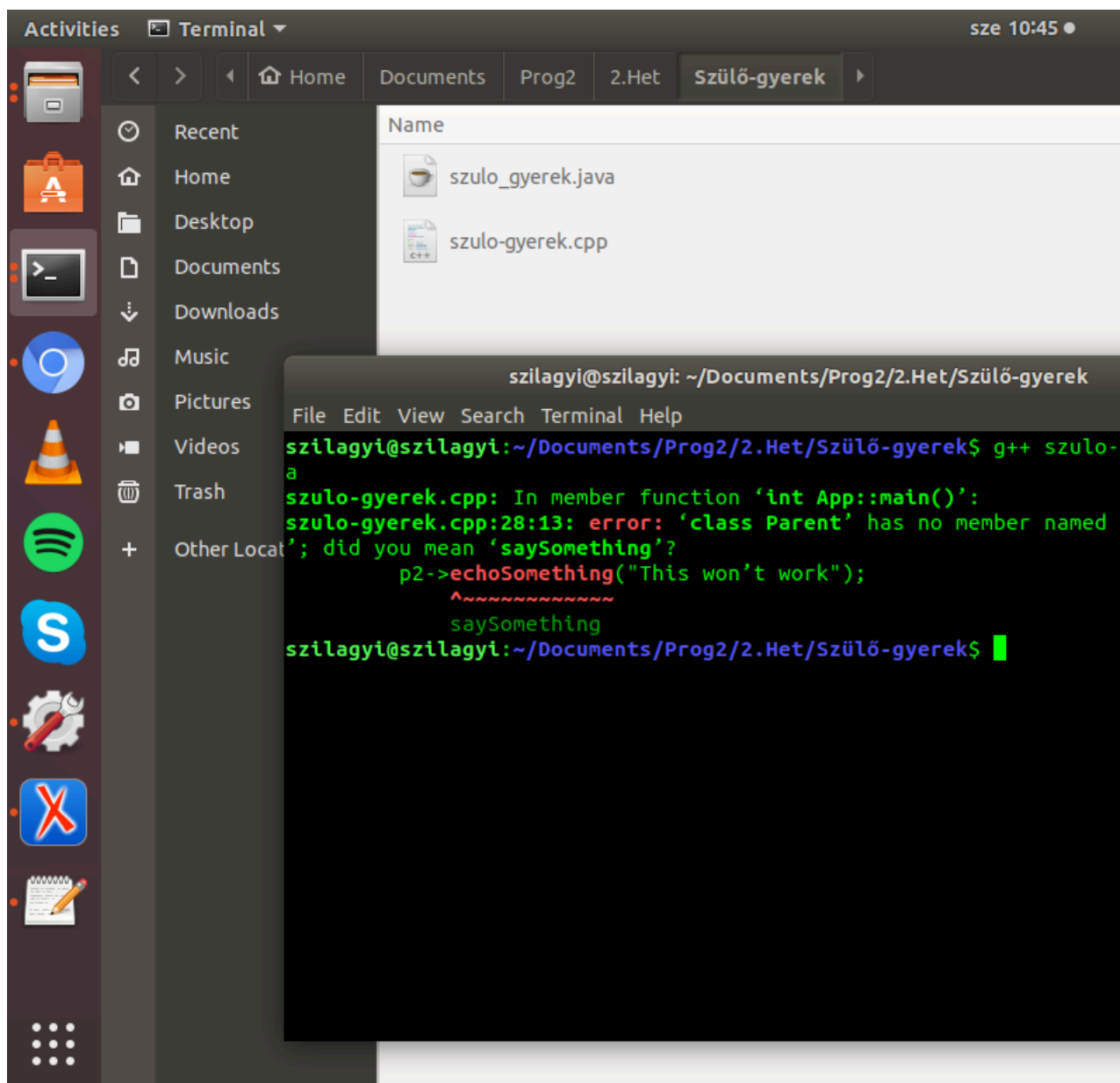
A nem Osök által definiált metódusokhoz nem férhetünk hozzá, hacsak nem # downcastoljuk az adott objektumot a tényleges típusára. Ez esetben viszont megsértjük az eloz# o feladatban ismertetett Liskov-elvet.

C++-ban:

```
#include <iostream>
#include <string>
class Parent
{
public:
    void saySomething()
    {
```



```
        std::cout << "Parent says: BLA BLA BLA\n";
    }
};
class Child : public Parent
{
public:
    void echoSomething(std::string msg)
    {
        std::cout << msg << "\n";
    }
};
class App
{
    int main()
    {
        Parent* p = new Parent();
        Parent* p2 = new Child();
        std::cout << "Invoking method of parent\n";
        p->saySomething();
        std::cout << "Invoking method of child through parent ref\n";
        p2->echoSomething("This won't work");
        delete p;
        delete p2;
    }
};
```



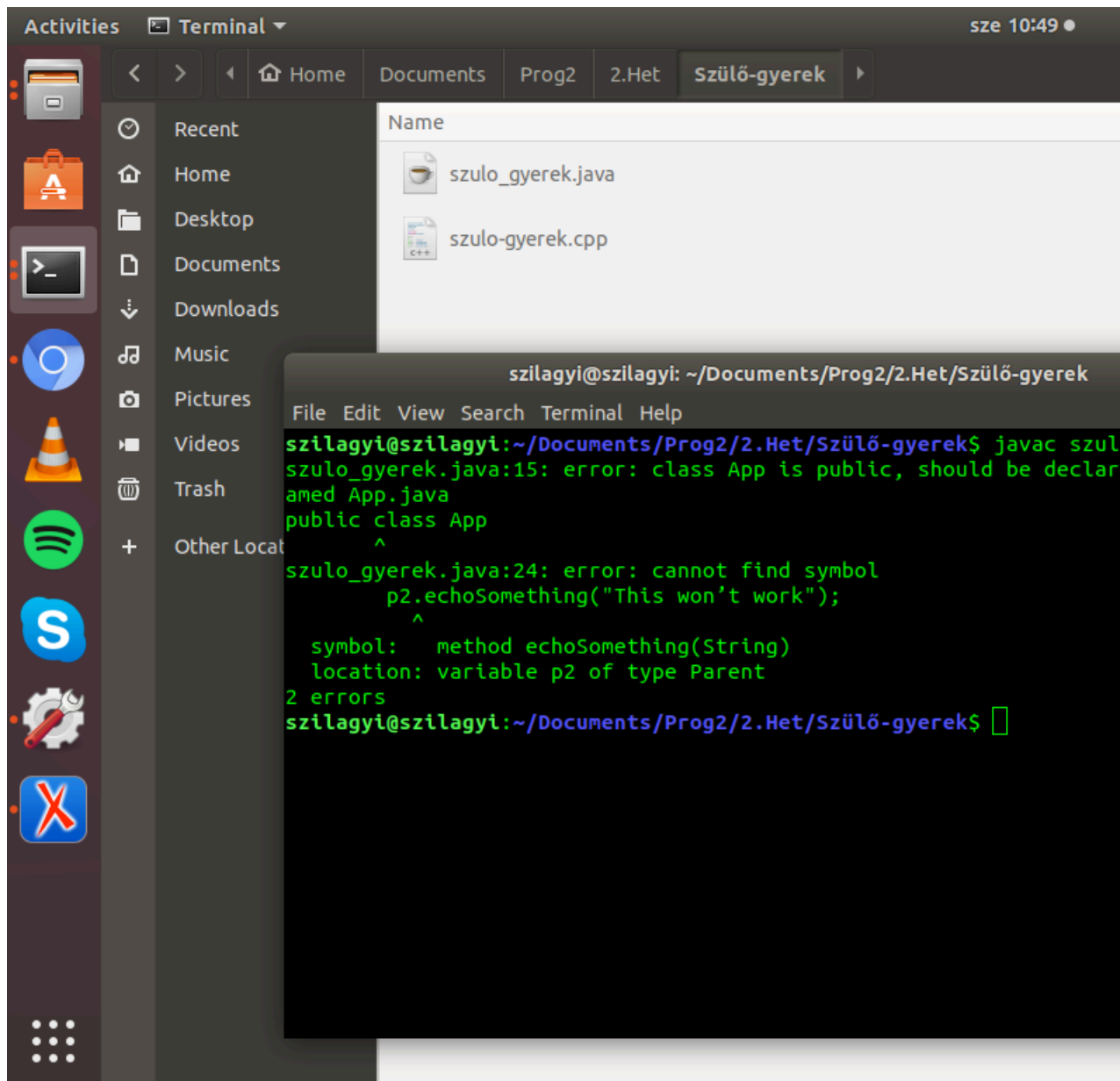
Javában:

```
class Parent
{
    public void saySomething()
    {
        System.out.println("Parent says: BLA BLA BLA");
    }
}
class Child extends Parent
{
    public void echoSomething(String msg)
    {
        System.out.println(msg);
    }
}
public class App
{
```

```

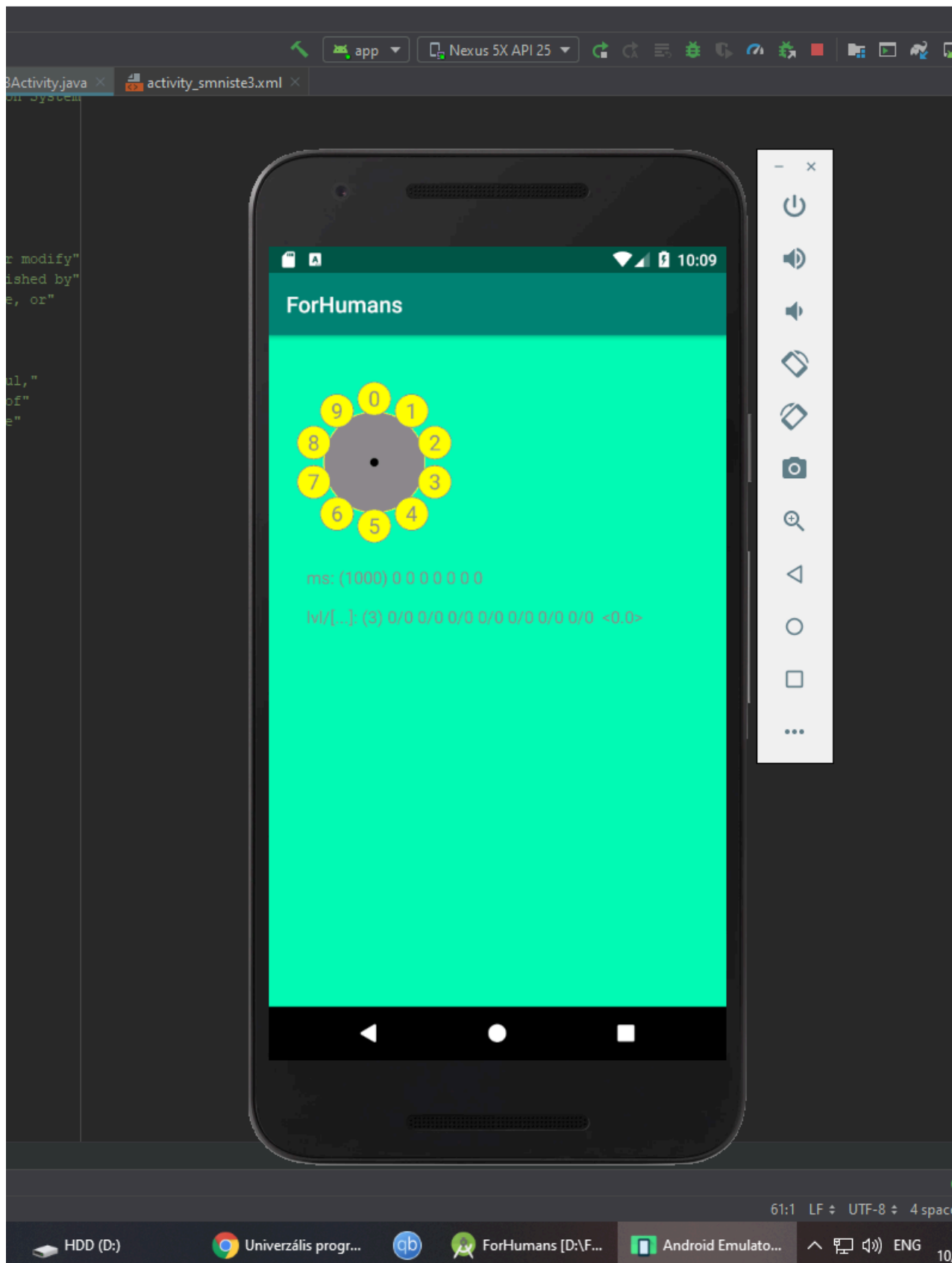
public static void main(String[] args)
{
    Parent p = new Parent();
    Parent p2 = new Child();
    System.out.println("Invoking method of parent");
    p.saySomething();
    System.out.println("Invoking method of child through parent ref");
    p2.echoSomething("This won't work");
}
}

```



Hello, Android!

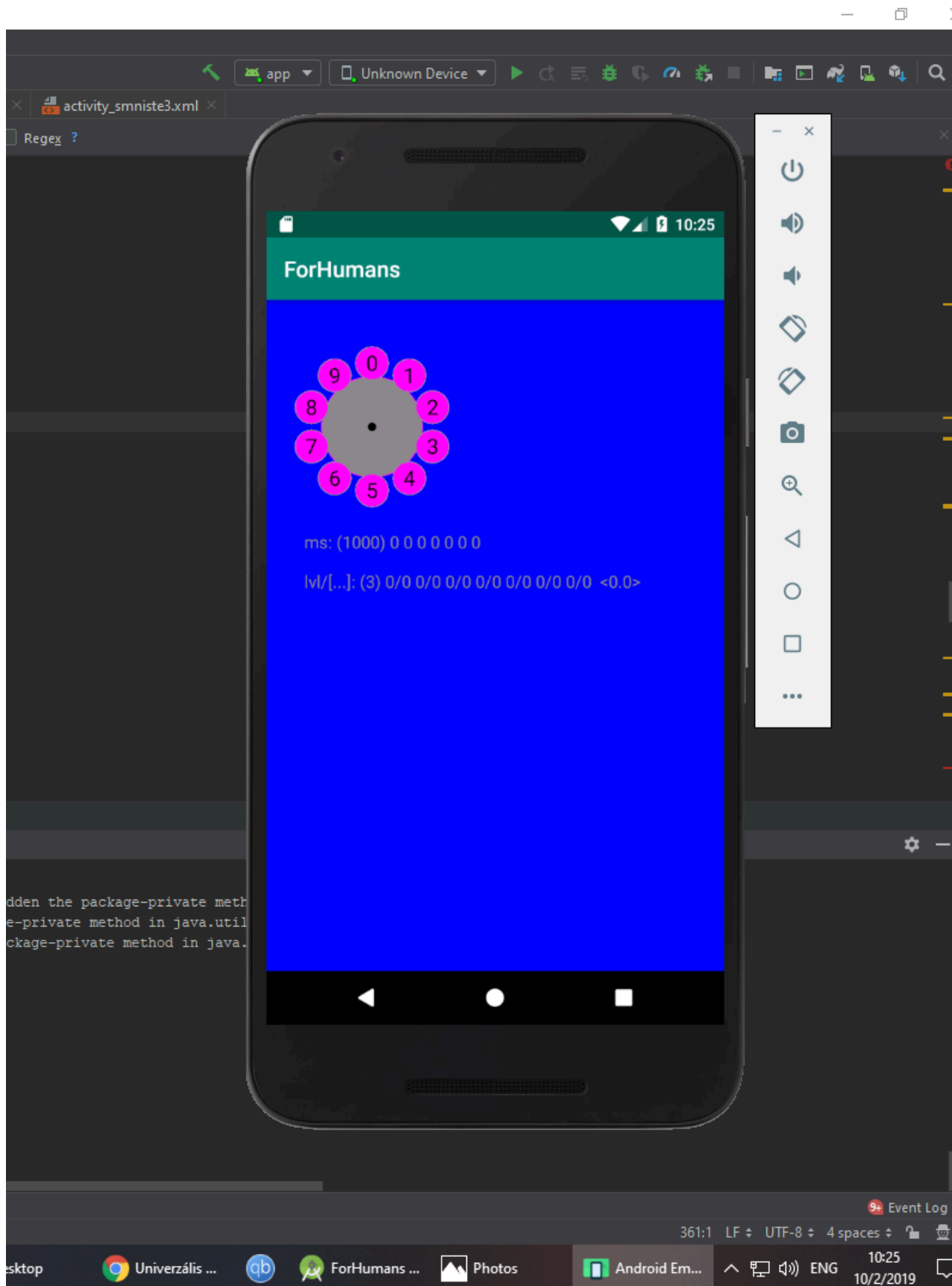
Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.



Márcsak a színvilágot kell megváltoztatni a feladat szerint: Ezt megtehetjük a SMNISTSurfaceView.java állományban a megfelelo#

- bgColor
- textPaint
- msgPaint
- dotPaint
- borderPaint
- fillPaint

s a többi változók értékeinek megváltoztatásával.



Chapter 5. 3. hét - „Helló, Mandelbrot!”

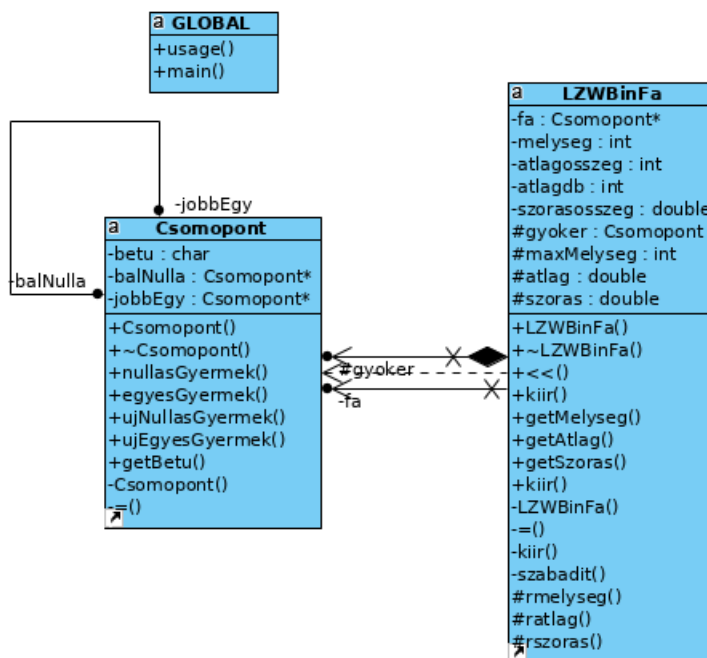
3. hét Modellez# eszközök és nyelvek. AZ UML és az UML osztálydiagramja.

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 kapcsolatra a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf (28-32 fólia)

A megoldás forrása:



A feladat megoldása során a Visual Paradigm programot használtam a kódból való uml diagram elkészítésére. Az uml diagrammok lényege hogy egy ábrában megmutassa egy adott program tervezetét. Ez kiválóan használható arra, hogy megtervezzünk egy programot, majd a diagramm alapján elkészítsük annak kódbéli verzióját. Ezt lehet úgy is hogy a terv alapján kézzel megírjuk a kódot vagy az ábra alapján le is lehet generálni a kódot. D

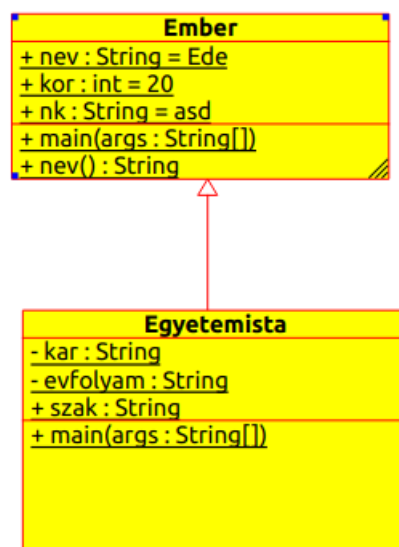
Ebben a feladatban az LZWBInfa kódból lett elkészítve a diagramm. A diagrammon látható hogy milyen osztályokból épül fel a program és azok között milyen kapcsolat van. Továbbá látható az is hogy az osztályok milyen változókkal és metódusokkal rendelkeznek. Mint látható egy osztály két részre van osztva a diagrammon. A felső részben található a változók listája, az alsó részben pedig a

metódusok listája. To- # vábbá minden változó és metódus előtt látható egy +, - vagy # jel. Ez jelöli azt hogy a hozzáférhet # osége az # milyen. A + jelenti a public-ot, a - a private-ot, a # pedig a protected jelzot jelenti.

Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generálunk bel#le forrást!

Ebben a feladatban egy UML diagramból kell kódot készíteni. Az UML diagrammot a Visual Paradigm programmal készítjük el majd utána kódot generálunk belöle. Els # onek el kell készíteni az osztályokat és # belehelyezni a kívánt változókat és metódusokat. Ha ez megvan akkor be kell jelölni a közöttük lévo# kapcsolatokat. Ez után már nincs más dolgunk mint elvégezni a kód generálást.



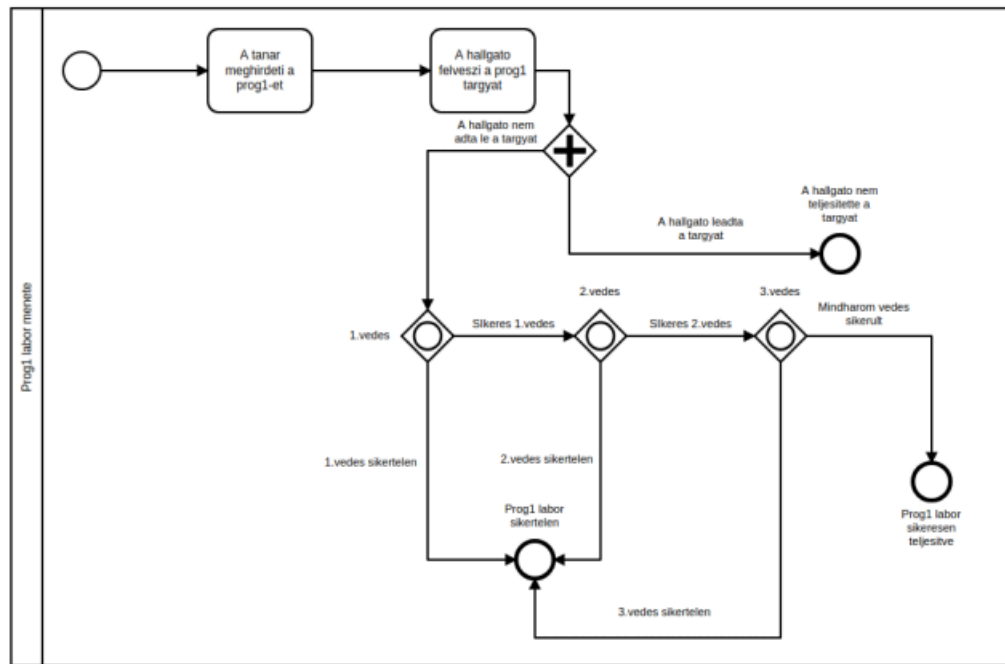
BPMN

Rajzoljunk le egy tevékenységet BPMN-ben!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_7.pdf (34-47 fólia)

A BPMN - teljes nevén Business Process Model and Notation - egy olyan fajta modellézt takar ami kiválóan alkalmas arra hogy egy folyamat lépéseit szemléltessünk benne. Be lehet vele mutatni hogy egy adott folyamat során milyen lépési lehetőségek vannak és hogy hová vezethetnek. Kiválóan alkalmas folyamatok # megtervezésére és arra hogy a folyamatok minden irányú kimenetelét szemléltessük.

Megoldás:



Jelen ábrán egy prog1 labor folyamata látható. A kezdő állapotot a vékony kör jelöli. # Ezután a folyamat lépéseket a téglalapok jelölik a nyilak pedig a folyamat irányát. A rombusz az elágazásokat jelenti ahonnan a kimeneteltől függően halad tovább a folyamat. A végén pedig a vastag kör a végállapotot jelenti ahol a folyamat véget ér.

Chapter 6. 4. hét - „Helló, Chomsky!”

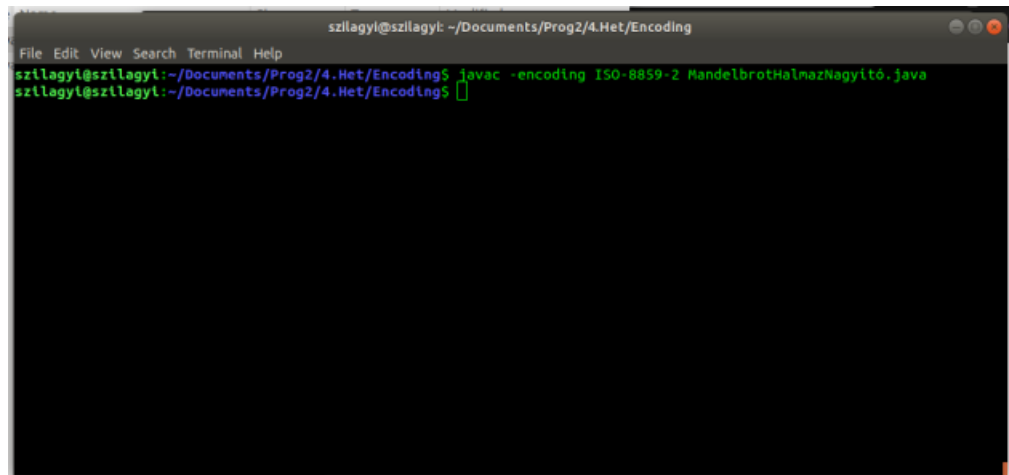
4. hét Objektumorientált programozási nyelvek programnyelvi elemei: karakterkészlet, lexikális egységek, kifejezések, utasítások.

Encoding

Fordítsuk le és futtassuk a Javat tanítók könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl neveiben és a forrásokban is meghagyjuk az ékezetes betűket!

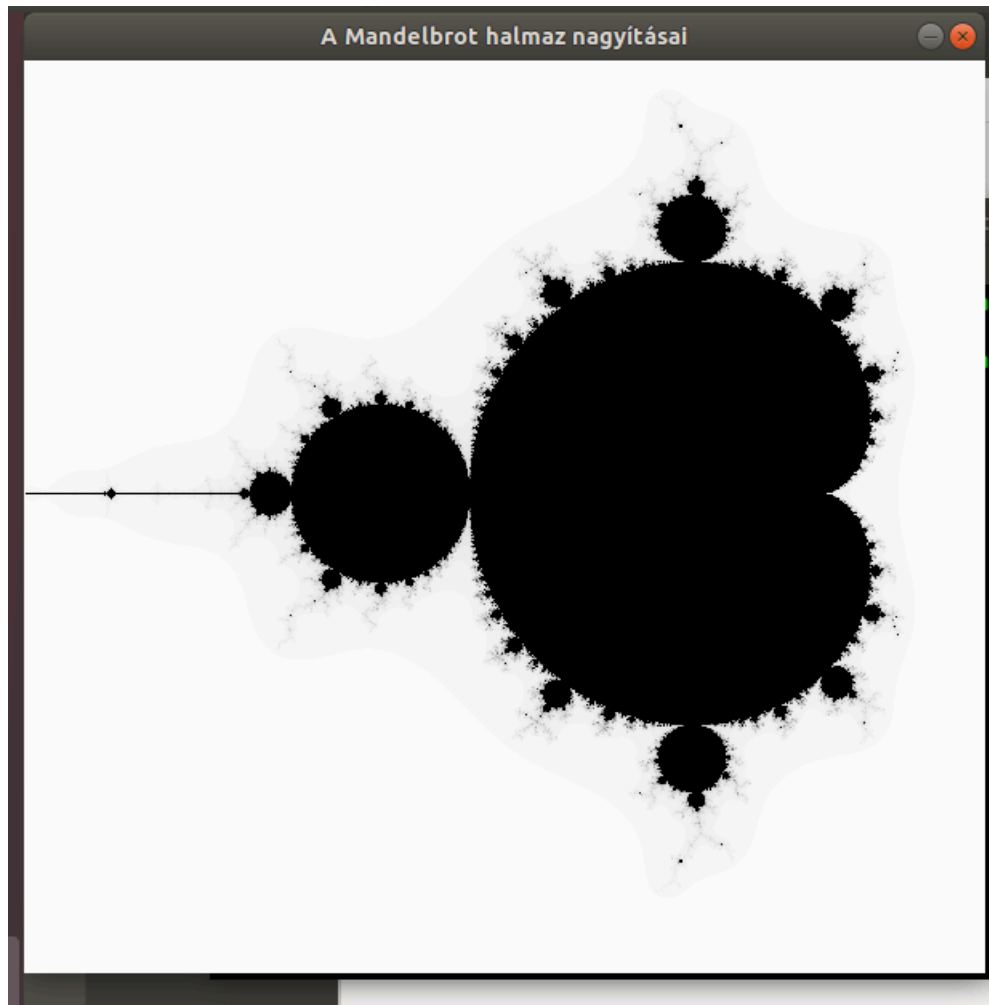
<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

A feladat során a MandelbrotHalmazNagyító.java fájlt kellett fordítani de úgy hogy az ékezetes karakterekkel együtt is forduljon. Ez úgy kivitelezhető ha a program fordítása során beadunk egy encoding kapcsolót a parancssori paraméterek közé: `javac -encoding "ISO-8859-2" MandelbrotIterációk.java` MandelbrotHalmazNagyító.java.



```
szilagyi@szilagyi: ~/Documents/Prog2/4.Het/Encoding
File Edit View Search Terminal Help
szilagyi@szilagyi:~/Documents/Prog2/4.Het/Encoding$ javac -encoding ISO-8859-2 MandelbrotHalmazNagyító.java
szilagyi@szilagyi:~/Documents/Prog2/4.Het/Encoding$
```

Az encoding kulcsszó segítségével adjuk meg a karakterkódolást amely segítségével értelmezni akarjuk a kódot. Jelen helyzetben ez a karakterkészlet az ISO-8859-2. Ez egy szabvány amely tartalmazza az ékezetes karaktereket és így ennek segítségével már működni fog a program.



I334d1c4

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a bet# helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez els# részben nem tette meg, akkor írasd ki és magyarázd meg a használt struktúrák memórafoglalását!)

A leet cipher alapjának írunk kell egy "ábécét", amiből majd a cipherünk válogatni tud. Ehhez használni tudjuk a fenti linket, ebben találunk pár példát a 1337 "ábécére".

```
import java.util.Scanner;

public class leetCipher
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        String text = in.next();

        for (int i = 0; i < text.length(); i++)
        {
            switch (text.charAt(i))
            {
                case 'a': System.out.print("4"); break;
                case 'b': System.out.print("8"); break;
                case 'c': System.out.print("("); break;
```

```

        case 'd': System.out.print("|}"); break;
        case 'e': System.out.print("3"); break;
        case 'f': System.out.print("|="); break;
        case 'g': System.out.print("6"); break;
        case 'h': System.out.print("|-|"); break;
        case 'i': System.out.print("1"); break;
        case 'j': System.out.print("|"); break;
        case 'k': System.out.print("|<"); break;
        case 'l': System.out.print("|_"); break;
        case 'm': System.out.print("(V)"); break;
        case 'n': System.out.print("/\\\/"); break;
        case 'o': System.out.print("0"); break;
        case 'p': System.out.print("|D"); break;
        case 'q': System.out.print("9"); break;
        case 'r': System.out.print("|2"); break;
        case 's': System.out.print("$"); break;
        case 't': System.out.print("7"); break;
        case 'u': System.out.print("|_|"); break;
        case 'v': System.out.print("\\\/"); break;
        case 'w': System.out.print("\\\/\\\/"); break;
        case 'x': System.out.print(")("); break;
        case 'y': System.out.print("y"); break;
        case 'z': System.out.print("2"); break;
        default: System.out.print(text.charAt(i)); break;
    }

    }
    System.out.println();
}
}

```

Lefuttatva:

```

szilagyi@szilagyi: ~/Documents/Prog2/4.Het/LeetChiper
File Edit View Search Terminal Help
szilagyi@szilagyi:~/Documents/Prog2/4.Het/LeetChiper$ javac leetCIPHER.java
szilagyi@szilagyi:~/Documents/Prog2/4.Het/LeetChiper$ java leetCIPHER
Hello
H3|_|_0
szilagyi@szilagyi:~/Documents/Prog2/4.Het/LeetChiper$ java leetCIPHER
world
\\\/0|2|_|}
szilagyi@szilagyi:~/Documents/Prog2/4.Het/LeetChiper$ 

```

Fullscreen

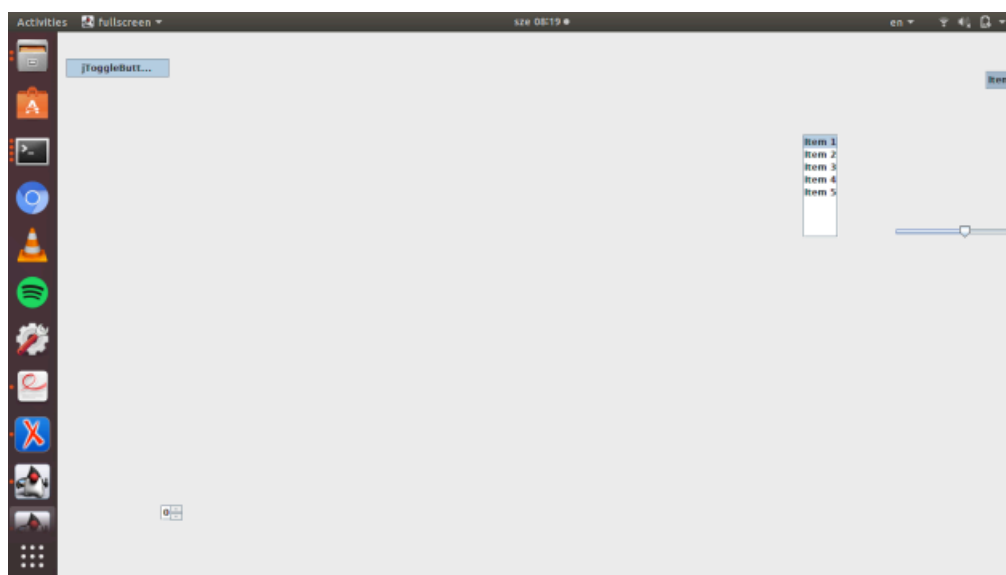
Készítsünk egy teljes képernyős Java programot!

Tipp:https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

A feladat megoldásához saját egyszeri programkódot használtam ami abból áll, hogy megjelenít a képernyőre különböző gombokat, csúszkákat. A teljes képernyős mód kódját a fullscreen.java fájlban tudjuk megnézni, a fullscreen metóduson belül:

```
public fullscreen() {  
    setUndecorated(true);  
    setAlwaysOnTop(true);  
    setResizable(false);  
    setVisible(true);  
    Toolkit tk= Toolkit.getDefaultToolkit();  
  
    int x=(int) tk.getScreenSize().getWidth();  
    int y=(int) tk.getScreenSize().getHeight();  
  
    setSize(x, y);  
    initComponents();  
}
```

Íme a lefutott teljesképernyős program:



Chapter 7. 5. hét - „Helló, Stroustrup!”

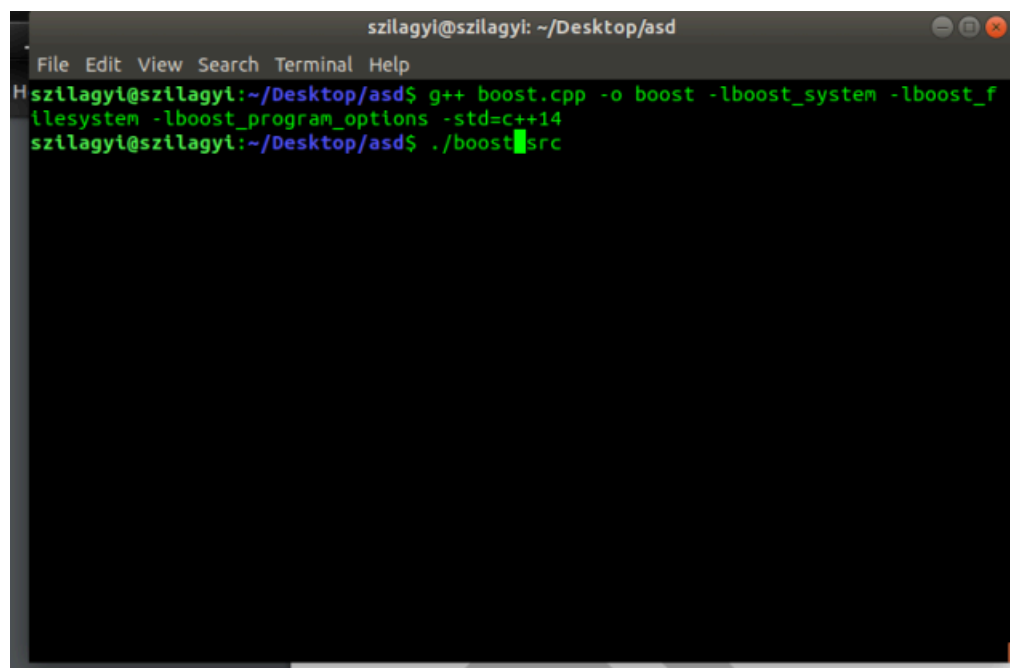
5. hét Objektumorientált programozási nyelvek típusrendszere (pl.: Java, C#) és 6. hét Típusok tagjai: mezők, (nevesített) konstansok, tulajdonságok, metódusok, események, operátorok, indexelők, konstruktorok, destruktorok, beágyazott típusok.Összevonva.

JDK osztályok

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

Források:<https://github.com/edeszilagyi/Prog2/tree/master/5.het/JDK%20oszt%C3%A1lyok>

A JDK program elkészítéséhez mintául szolgálhat a fénykard.cpp fájl, melynek függvénye a fájlok kereséséért és listázásáért felelős. A boost.cpp-val tehát a JDK-ban található src.zip-ben lévő java állományokat fogjuk kilistázni a könyvtárszerkezet kicsomagolása után, majd ezek alapján kiíratjuk a JDK osztályok számát. A Boost könyvtár segítségével rekurzívan megyünk végig a könyvtárszerkezeten, így a java fájlokban lévő osztályok kigyűjtése felgyorsul és leegyszerűsödik. Ha az src.zip tartalmát kicsomagoltuk, akkor következhet a .cpp fordítása. Ezután ha argumentumként megadjuk az src mappát utána futtatáskor megkapjuk a JDK osztályok számát. A kódok futtatása:



```
szilagyi@szilagyi: ~/Desktop/asd
File Edit View Search Terminal Help
H szilagyi@szilagyi:~/Desktop/asd$ g++ boost.cpp -o boost -lboost_system -lboost_filesystem -lboost_program_options -std=c++14
szilagyi@szilagyi:~/Desktop/asd$ ./boost src
```

```

szilagyi@szilagyi: ~/Desktop/asd
File Edit View Search Terminal Help
"src/java.datatransfer/java/awt/datatransfer/Clipboard.java"
"src/java.datatransfer/java/awt/datatransfer/MimeTypeParseException.java"
"src/java.datatransfer/java/awt/datatransfer/FlavorEvent.java"
"src/java.datatransfer/java/awt/datatransfer/SystemFlavorMap.java"
"src/java.datatransfer/java/awt/datatransfer/DataFlavor.java"
"src/java.datatransfer/java/awt/datatransfer/ClipboardOwner.java"
"src/java.datatransfer/java/awt/datatransfer/MimeType.java"
"src/java.datatransfer/java/awt/datatransfer/FlavorListener.java"
"src/java.datatransfer/java/awt/datatransfer/MimeTypeParameterList.java"
"src/java.datatransfer/sun/datatransfer/DesktopDatatransferService.java"
"src/java.datatransfer/sun/datatransfer/DataFlavorUtil.java"
"src/java.datatransfer/module-info.java"
"src/jdk.net/jdk/net/NetworkPermission.java"
"src/jdk.net/jdk/net/ExtendedSocketOptions.java"
"src/jdk.net/jdk/net/SocketFlow.java"
"src/jdk.net/jdk/net/package-info.java"
"src/jdk.net/jdk/net/LinuxSocketOptions.java"
"src/jdk.net/jdk/net/Sockets.java"
"src/jdk.net/jdk/nio/package-info.java"
"src/jdk.net/jdk/nio/Channels.java"
"src/jdk.net/module-info.java"
"src/jdk.jdwp.agent/module-info.java"
18294
szilagyi@szilagyi:~/Desktop/asd$

```

Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

A megoldás forrása: <https://github.com/edeszilagyi/Prog2/tree/master/5.het/Változó%20argumentumszámú%20ctor>

A kétdimenziós halmaz png ábráját fogjuk eloször létrehozni a mandel.cpp file segítségével. hogy látjuk, a mandel.cpp sikeres fordításához és futtatásához szükségünk lesz a libpng, libpng++ könyvtárakra és a png++/png.hpp fájlra. A könyvtárakat a sudo parancs segítségével tudjuk telepíteni, viszont a headerhez le kell töltenünk a png++ hivatalos oldaláról a becsomagolt telepítő állományokat: <http://download.savannah.nongnu.org/releases/pngpp/png++-0.2.9.tar.gz>

```

//A terminálba leadott parancsok:
$ sudo apt-get install libpng-dev
$ sudo apt-get install libpng++-dev
$ tar -zxf png++-0.2.9.tar.gz
$ cd ./png++-0.2.9
$ make

```

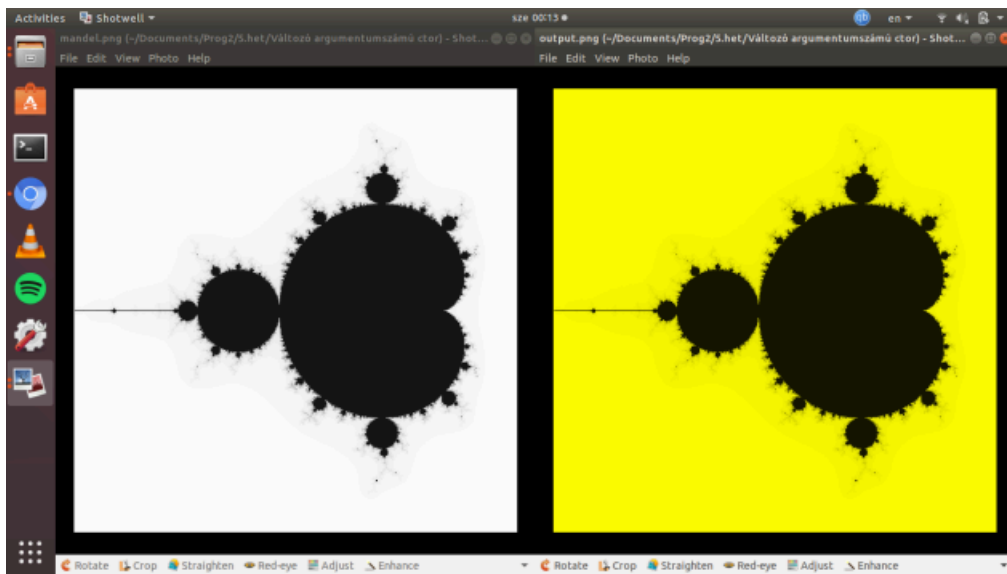
Ezután lehet fordítani és futtatni a programot:

```

szilagyi@szilagyi: ~/Documents/Prog2/5.het/Változó argumentumszámú ctor
File Edit View Search Terminal Help
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$ g++ mandel.cpp `libpng-config --ldflags` -o mandel
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$ ./mandel mandel.png
Számítás.....
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$ g++ main.cpp nlp.hpp -o a -lpng
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$ ./a mandel.png
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$
szilagyi@szilagyi:~/Documents/Prog2/5.het/Változó argumentumszámú ctor$

```

A perceptron az egyik legegyszerűbb elreccsatolt neurális hálózat. A main.cpp segítségével fogjuk szimulálni a hiba-visszaterjesztési módszert, mely a többretegű perceptronok egyik legfőbb tanítási módszere. Ahhoz, hogy ezt fordítani és futtatni tudjuk később, szükségünk lesz az nlp.hpp fájlra, mely már tartalmazza a Perceptron osztályt. Az előző program futtatásával létrejött Mandelbrot png ábrát fogjuk beimportálni. A fájloknak köszönhetően megadhatjuk a neuronok darabszámát. A mandel.png alapján új képet állítunk elő. A visszakapott értékeket megfeleltetjük a blue értékeknek. A 4. heti Perceptron feladatához képest módosításokat kell végeznünk a header file-on is, ugyanis új képet akarunk előállítani. Az operátor már egy tömböt térít vissza, melynek segítségével bele tudunk nyúlni a képbe.



Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutatást „esszé szöveget”

Másoló-mozgató szemantika

1. Másoló szemantika:

Ha van az osztályban dinamikus adattag, akkor alapértelmezésben tiltjuk, míg privátban deklaráljuk a másoló konstruktort és másoló értékadást. Különböztetjük a “Rule of three” szabályt.

A Rule of three szabály kimondja, hogy ha dinamikus tagot aggregáló osztályban kell implementálni:

- Destruktort,

- Másoló konstruktort,
- Vagy másoló értékadást

Akkor nagy valószínűséggel mindhármát implementálni kell.

Példa Rule of three-re:

```
Int(const Int&);  
Int(Int&&); //Rule of 5  
Int& operator=(const Int&);  
Int& operator=(Int&&);  
~Int();
```

2.Mozgató szemantika:

C++11-be bekerült a mozgató konstruktor, amely alkalmazása drasztikusan hatékonyabbá tette a kódokat. A mozgató konstruktor legfontosabb tulajdonsága, hogy a használatával elkerülhetők a memória újraelosztása és ugyanannyi memóriát használ mint amennyit az eredeti objektum használt, aminek ideiglenes funkciója van, mivel miután átadta az értéket az új objektumnak törlésre kerül. Ahhoz, hogy alkalmazni tudjuk a mozgató szemantikát, először meg kell oldanunk azt a problémát, hogy az eredeti objektumot hibamentesen töröljük.

Példa:

```
#include<bits/stdc++.h>  
int main()  
{  
    std::vector<int> vec1 {1, 2, 3, 4, 5};  
    std::vector<int> vec2 {6, 6, 6, 6, 6};  
  
    std::cout << "1.vektor elemei :";  
    for(int i = 0; i < vec1.size(); i++)  
        std::cout << " " << vec1[i];  
    std::cout << "\n";  
  
    std::cout << "2.vektor elemei :";  
    for(unsigned int i = 0; i < vec2.size(); i++)  
        std::cout << " " << vec2[i];  
    std::cout << "\n\n";  
  
    // elso 3 elemet az 1.vektorból move-olja a 2.vektor 1.elemetől  
    std::move(vec1.begin(), vec1.begin() + 3, vec2.begin());  
  
    std::cout << "2.vektor elemei std::move után:";  
    for(unsigned int i = 0; i < vec2.size(); i++)  
        std::cout << " " << vec2[i];  
    std::cout << "\n";  
  
    return 0;  
}
```

Lefuttatva:

```

szilagyi@szilagyi: ~/prog2/5.het/Összefoglaló
File Edit View Search Terminal Help
szilagyi@szilagyi:~/prog2/5.het/összefoglaló$ g++ move.cpp -o a
szilagyi@szilagyi:~/prog2/5.het/összefoglaló$ ./a
1.vektor elemei : 1 2 3 4 5
2.vektor elemei : 6 6 6 6 6

2.vektor elemei std::move után: 1 2 3 6 6
szilagyi@szilagyi:~/prog2/5.het/összefoglaló$ 

```

Table 7.1. Copy ctor vs Move ctor

Másoló konstruktor	Mozgató konstruktor
Az argumentum referenciaként a bal oldali értéket kapja.	Az argumentum referenciaként a jobb oldali értéket kapja.
Új objektumot hoz létre a kapott objektumból az által, hogy lemásolja az összes értékét egy új memória címre.	Új objektumot hoz létre, de ugyanannyi memóriát használ mint az átadott objektum.
Nem hatékony, mivel sok memória elosztást használ m#kódése során az új objektumok létrehozására.A mozgató konstruktor használata sokkal hatékonyabb.	Memóriahasználatának nagy részét a kapott objektum teszi ki, ezért hatékonyabb, mint a másoló konstruktor.
Nem változtat semmit a kapott objektumon, ezért a kapott objektum használható másoló operátor után is.	Mivel a mozgató konstruktor a kapott objektum memória blokkjait használja, ezért a kapott objektum nem használható mozgató operátor után.

Chapter 8. 6. hét - „Helló, Gödel!”

7. hét Interfészek. Kollektiók. és 8. hét Funkcionális nyelvi elemek. Lambda kifejezések.Összevonva.

STL map érték szerinti rendezése

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

Források:<https://github.com/edeszilagi/Prog2/tree/master/6.het/stl%20map>

Ebben a példában megismerjük a C++ STL map adatszerkezetét. Ez az adatszerkezet úgynevezett párok tárolására képes, általában a párok első elemét hívjuk kulcsnak, a másodikat pedig értéknek. Új értékeket az insert() metódus meghívásával lehet hozzáadni, mely paramétereként elfogad egy std::pair típusú objektumot. A mapoknak két tagjuk van (most string és int), ezeket rendeljük hozzá egy pair vektorhoz, és ezt követően rendezünk. Érték szerint csökkenő sorrendben történik a rendezés (p1.second és p2.second használatával). Lambda kifejezés segítségével rendezünk: a nagyobb elemek lesznek első, ugyanis ha az első pair érték nagyobb, akkor igazzal tér vissza a függvény.

```
std::vector<std::pair<std::string, int>> sort_map ( std::map <std::string, int>
{
    std::vector<std::pair<std::string, int>> ordered;

    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [ = ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        }
    );

    return ordered;
}
```

Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable<T> szerepét!

Források:<https://github.com/edeszilagi/Prog2/tree/master/6.het/alternativ%20tabella>

Az alternatív tabella a Google algoritmusán, a PageRank-en alapszik. A PageRank ötlete, hogy azok a weblapok jobb minőségűek, amelyekre jobb minőségű lapok mutatnak. Jelen esetben az foglalja el a legjobb helyet a labdarúgó bajnokságon, aki a legjobb helyen lévő csapatoktól szerez pontot. A táblázatban lévő eredményeket a kereszt és kereszt1 nevű kétdimenziós tömbökbe dobjuk bele a Wiki2Matrix osztály-ban. Azért kell még egy kereszt1nevű táblázat mert a magyar bajnokságban mindenki 3 alkalommal játszik mindenkivel. Vegyük például a 2018-19-es magyar labdarúgó bajnokság első osztályának eredményeit. Az üres legyen 0, a győzelem 1, a döntetlen 2, a vereség pedig 3. A példa alapján a mátrixokba tehát a következők értékek kerülnek:

```

int[][] kereszt = {
    {0, 1, 1, 3, 1, 1, 2, 1, 1, 1, 2, 3},
    {1, 0, 1, 1, 2, 1, 2, 2, 1, 1, 2, 3},
    {1, 1, 0, 3, 1, 2, 2, 1, 2, 2, 3, 3},
    {1, 2, 1, 0, 1, 1, 1, 1, 2, 1, 1, 2},
    {3, 3, 2, 1, 0, 3, 3, 3, 3, 1, 2, 3},
    {3, 1, 2, 3, 1, 0, 3, 1, 2, 1, 3, 2},
    {3, 2, 1, 3, 1, 2, 0, 3, 1, 1, 2, 1},
    {2, 3, 1, 3, 1, 3, 2, 0, 3, 1, 1, 3},
    {2, 1, 3, 3, 2, 1, 1, 1, 0, 1, 2, 3},
    {1, 3, 1, 1, 1, 2, 1, 3, 2, 0, 3, 1},
    {2, 1, 1, 2, 1, 1, 2, 3, 2, 1, 0, 1},
    {1, 2, 3, 1, 1, 1, 1, 3, 2, 3, 1, 0}
};

int[][] kereszt2 = {
    {0, 2, 3, 1, 3, 0, 0, 0, 1, 0, 0, 3},
    {0, 0, 1, 0, 1, 3, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 3, 0, 0, 0, 1, 1, 1, 1},
    {0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1, 2},
    {2, 0, 2, 3, 3, 0, 1, 0, 0, 0, 0, 0},
    {1, 0, 1, 3, 1, 0, 0, 0, 1, 0, 0, 0},
    {3, 0, 1, 3, 0, 3, 3, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 3, 2},
    {2, 1, 0, 2, 0, 3, 1, 1, 0, 0, 0, 0},
    {2, 2, 0, 0, 0, 1, 1, 2, 0, 3, 0, 0},
    {0, 1, 0, 0, 0, 1, 1, 1, 0, 2, 1, 0}
};

```

A fordítás és futtatás után kapott új értékeket az AlternatívTabella osztályba kell belehelyezni, a `double[][] Lnk = {}` rész kapcsos zárójelei közé. Az `csapatNevE` tömbbe kerülnek a csapatok nevei a táblázatban való megjelenésük alapján. Az `ep` tömbbe az eredeti tabella pontjait írjuk be. Készítünk egy `csapatNevL` tömböt is, melybe a Wiki2Matrix `kereszt` nev# mátrixa alapján létrejött sorrend szerint kerülnek be a csapatok.

Az alternatív tabella értékeinek összehasonlításához, rendezéséhez igénybe vesszük a `java.lang` package `Comparable` interfészét. Az interface lehetővé teszi, hogy képessé legyünk alkalmazni a listákra és tömbökre definiált függvényeket. A `compareTo` metódus a paraméterként megadott objektumot hasonlítja össze az aktuális objektummal. Háromféle értéket adhat vissza: pozitív, negatív számot vagy nullát.

```

class Csapat implements Comparable<Csapat> {

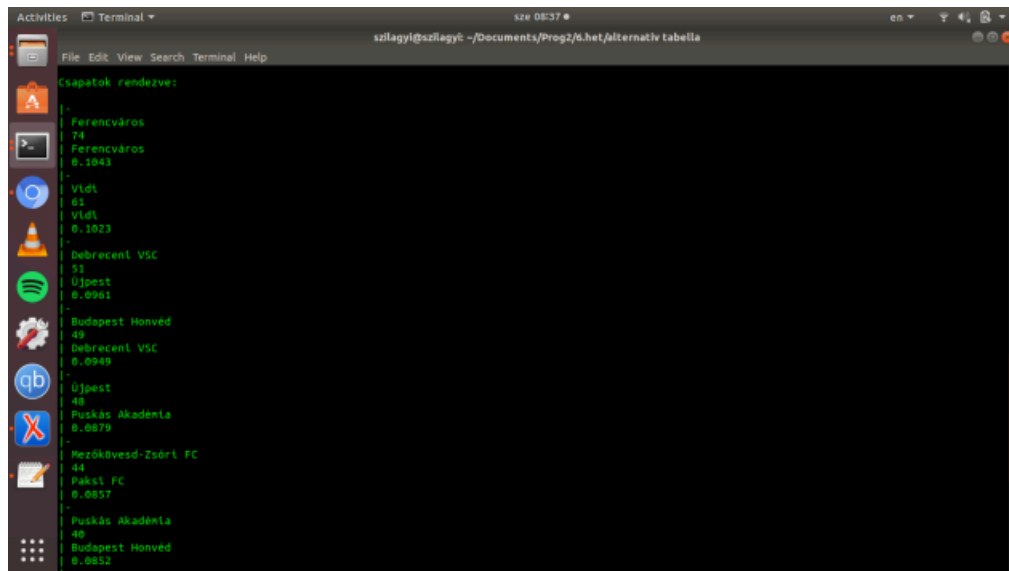
    protected String nev;
    protected double ertek;

    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;
        }
    }
}

```

```
}
}
```



Prolog családfa

Ággyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

Források: <https://github.com/edeszilagyi/Prog2/tree/master/6.het/Prolog>

A Prolog egy olyan programozási nyelv, melynek segítségével matematikai logikai formulákat tudunk vizsgálni. A matematikai logikáról ez a feladat nem fog részletesebb tájékoztatást nyújtani, a fentebb lévő pdf-ben találhatók példákat a mélyebb megértéshez. Annyit fontos megemlíteni, hogy a program az elsőrendű logikán alapszik. Tehát itt az atomi formulák mellett megjelennek a függvényszimbólumok és a kvantorok, mint a logikai formulák építőelemei.

A programunk Java programban íródott, mely az SWI-Prolog könyvtárat használja. Alapvetően a program úgy működik, hogy az elsőnek betöltjük a Prolog fájlt, majd arról készítünk lekérdezéseket. A family.pl fájl tartalmazza a pdf-ben megadott Prolog programot:

```
férfi(nándi).
férfi(matyi).
férfi(norbi).
férfi(dodi).
férfi(joska).
nő(gréta).
nő(erika).
nő(kitti).
nő(marica).
gyereke(nándi, norbi).
gyereke(matyi, norbi).
gyereke(gréta, norbi).
gyereke(nándi, erika).
gyereke(matyi, erika).
gyereke(gréta, erika).
gyereke(norbi, dodi).
gyereke(norbi, kitti).
gyereke(erika, joska).
gyereke(erika, marica).
apa(X) :- férfi(X), gyereke(_Y, X).
```

```

apja(X, Y) :- férfi(X), gyereke(Y, X).
anya(X) :- nő(X), gyereke(_Y, X).
anyja(X, Y) :- nő(X), gyereke(Y, X).
nagyapa(X) :- apja(X, Y), (apja(Y, _U); anyja(Y, _Z)).
nagyapja(X, Z) :- apja(X, Y), (apja(Y, Z); anyja(Y, Z)).

```

Ezt a programunkba a következő módon fogjuk beolvasni:

```

String s = "consult('family.pl')";
Query q = new Query(s);
System.out.println(q.hasSolution());

```

Itt jól látható, hogy hogyan fog m#ködni maga az alkalmazás. Els#nek megadunk egy stringet, mely kiértékelni kívánt formulát tartalmazza. Majd létrehozunk egy Query osztályú objektumot hozunk létre. Ez az osztály teszi lehetővé, hogy kiértékeljük a formula igazságértékét, vagy az egyes változók lehetséges értékeit. Ha csak az igazságértékre vagyunk kíváncsiak, akkor a hasSolution() függvényt kell használni. Ha itt hamis eredményt ad, akkor a fájl nem találja.

```

String t2 = "apa(gréta)";
System.out.println(t2 + " is " + (Query.hasSolution(t2) ? "provable" :

```

Az els# példa tehát azt mutatja be, hogy hogyan lehet megtudni a formula igazságértékét. Jelen esetben azt vizsgáljuk, hogy Gréta apának tekinthet#-e. Nyilván nem, vagyis a terminálban a "not provable" kifejezés # jelenik meg.

```

String t3 = "nagyapja(X, matyi)";
System.out.println("each solution of " + t3);
Query q3 = new Query(t3);
while (q3.hasMoreSolutions()) {
    Map<String, Term> s3 = q3.nextSolution();
    System.out.println("X = " + s3.get("X"));
}

```

A változók értékének lehetséges értékeit kétféleképpen lehet felsorolni. A fenti példában azokat az Xeket keressük, akik Matyi nagyapjai lehetnek. Ezt úgy tudjuk megfogalmazni matematikai logikával, hogy azokat az X-eket keressük, akik Matyi apjának vagy anyjának az apja. Tehát végig iterálunk a lehetséges X-eken.

Chapter 9. 7. hét - „Helló, !”

9. hét Adatfolyamok kezelése, streamek és 11. hét I/O, állománykezelés. Szerializáció.

FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon!

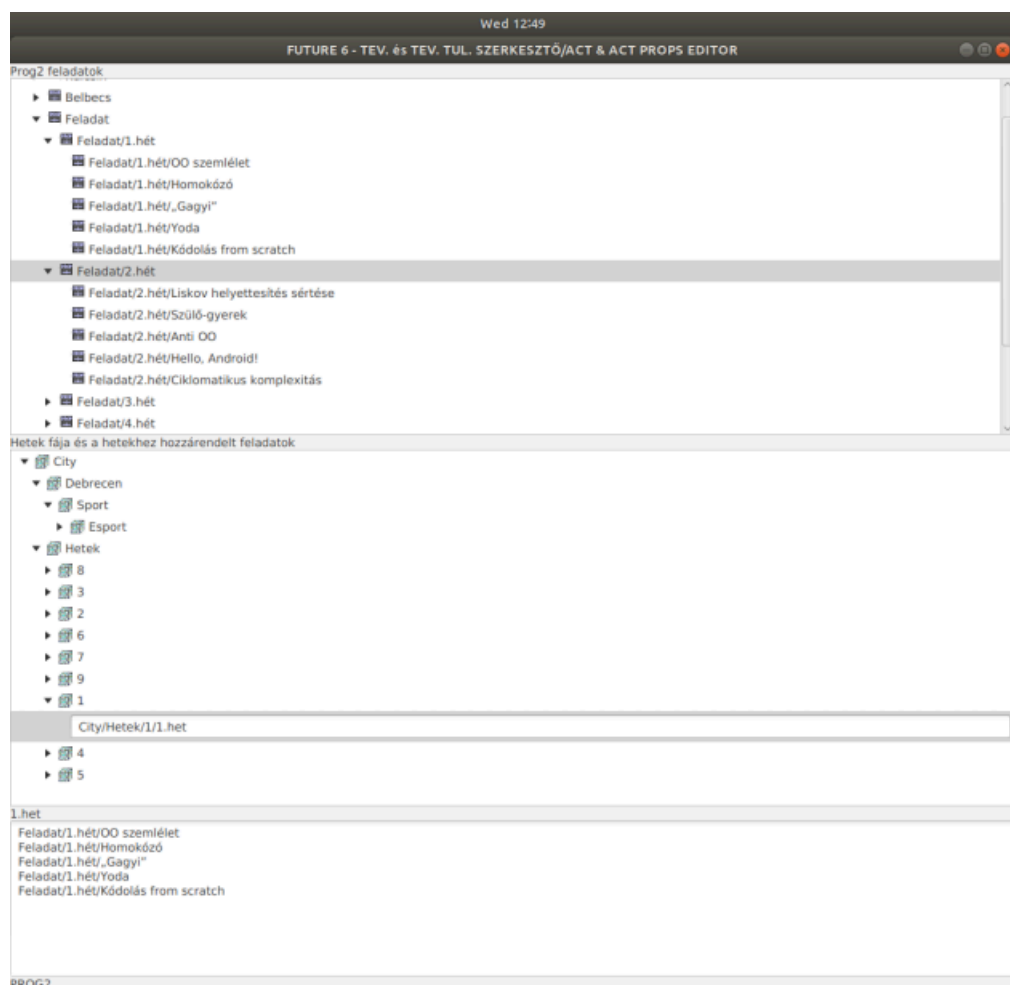
<https://github.com/nbatfai/future/tree/master/cs/F6>

Itt láthatjuk m#k#d#sben az alapot: <https://www.twitch.tv/videos/222879467>

A future projekt eredeti célja egy város alternatív j#voinek legenerálása, a legenerált j#v#k elemzése volt. A future6-hoz készült egy tulajdonság editor, melyben a hallgatók az adott nap tevékenységeit tudták feljegyezni.

A javításom a programon a lett hogy a f#bb címkék alpból ki vannak nyitva, így megkönnyítik a felhasználó dolgát a program használata közben, mivel így egb#l észreveszi azt a dolgot amit keres.

A program m#k#d#s közben:



OOOWC Boost ASIO hálózatkézelése

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

Scanf beépített függvény segítségével adatokat olvashatunk be, amelyet típus szerint eltérő módon kell megadni.

A scanf függvény argumentumában % jel után adjuk meg a beolvasott adat típusának megfelelő jelölést, amely nagyon sokféle lehet, de a legfontosabbak és a leggyakrabban használtak a következők:

- 1) %d: adatot decimális számmá alakít
- 2) %x: adatot hexadecimális számmá alakít
- 3) %f: adatot lebegőpontos számmá alakít
- 4) %s: adatot stringgé alakít

A carlexer.ll programban a scanf-nek az a szerepe, hogy különböző adatokat olvas be, amelyek segítségével megadhatóak az autók kezdőpozíciói, lehetséges útvonalai illetve az autó pozíciója, elhelyezkedése a program teljes működésénél.

BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esportal-ent-search>

Signal-slot mechanizmus:

A signalok és slotok biztosítják az objektumok közötti kommunikációt.

A signals and slots mechanizmus teszi egyedivé a Qt-t mivel ez teljes egészében Qt-specifikus, és ez a mechanizmus különbözteti meg más toolkitektől a Qt-t.

Míg más toolkit a callback függvénypointer segítségével értesítenek egy esemény bekövetkeztéről, addig a Qt-ben lehetőségünk van egy alternatív mechanizmust alkalmazni, ami a Slot-signal mechanizmus.

Signal

A Qt-ban az objektumok képesek magukból signalokat kiadni, sugározni (emit), ami akkor történik, ha az objektummal történt valami fontos, például megváltozott az állapota, a BrainB esetében ez akkor történik meg amikor rákattintunk a négyzetbeli pontra.

Csak az az osztály és az osztály alosztályai képesek signalt sugározni, amelyekben definiálva van a signal. Amikor sugároz a signal, akkor a slot egyből elvégződik, mint egy függvényhívás.

A signalok automatikusan generálódnak a moc által, ezért ezeket nem kell a .cpp fileban implementálni, és nincs return értékük.

Slot

A Qt-ban az objektumoknak lehetnek slotjaik is, amelyek olyan speciális tagfüggvények, melyek képesek érzékelni azt, ha egy másik objektumnak megváltozott az állapota.

Ha több slot van egy signálhoz kapcsolódva, akkor a slotok egymás után kerülnek elvégzésre. A slotok egyszerű C++ függvények, azzal a különbséggel, hogy ezekhez hozzá lehet kapcsolni signalokat. A slot argumentumoknak nem lehet alapértelmezett értékük, mindig saját egyedi értékeket adjunk nekik.

3 féle slot letezik:

1.Public slot: olyan slotokat tartalmaz, amelyekhez bárki kapcsolhat signalt, ez nagyon hasznos mert az olyan objektumok között is megoszthatunk információkat, amelyek egyébként nem tudnak egymásról semmit, de összekapcsoljuk a signaljaikat és slotjaikat.

2.Protected slot: olyan slotokat tartalmaz, amelyekhez ez az osztály illetve alosztályai kapcsolhatnak signalokat. Ez olyan slotoknak lehet hasznos, amelyeket ebben az osztályban implementáljuk.

3.Private slot: csak ez az osztály kapcsolhat hozzá signalt, ez olyan szorosan kapcsolt osztályok esetén hasznos, amelyekhez még az alosztályai sem kapcsolódhatnak.

Definiálhatjuk a slotokat virtualeként is.

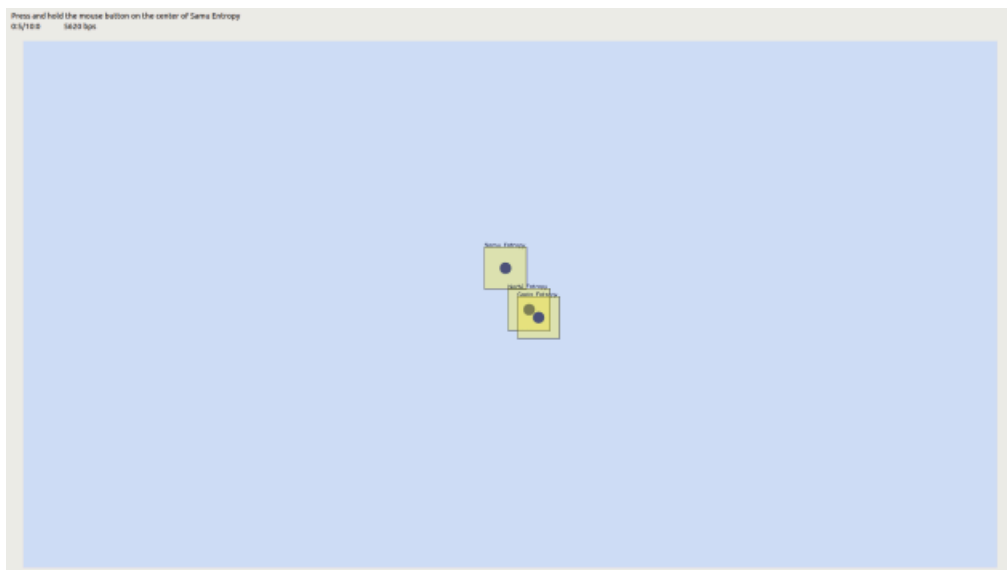
Amikor a signal-t a slot-hoz kötjük, tulajdonképp elmondjuk a slot-nak, hogy melyik objektum állapotváltozására kell figyelnie. Figyeljünk fel arra a tényre, hogy a megfigyelt

objektumnak fogalma sincs róla, hogy valaki figyel-e rá. A kapcsolatért, azért,

hogy történjen valami, lényegében a slot objektuma felel.

A signal-slot kapcsolatok miatt a számított adatok értéke bármely „számszer#” adat megváltozásakor újraszámolóódik.

És a programról kép m#kódése közben:



Chapter 10. 8. hét - „Helló, Lauda!”

10. hét Kivételkezelés. és 12. hét Reflexió. A fordítást és a kódgenerálást támogató nyelvi elemek (annotációk, attribútumok). Összevonva.

Port scan

Mutassunk rá ebben a port szkennel# forrásban a kivételkezelés szerepére!

<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id527287>

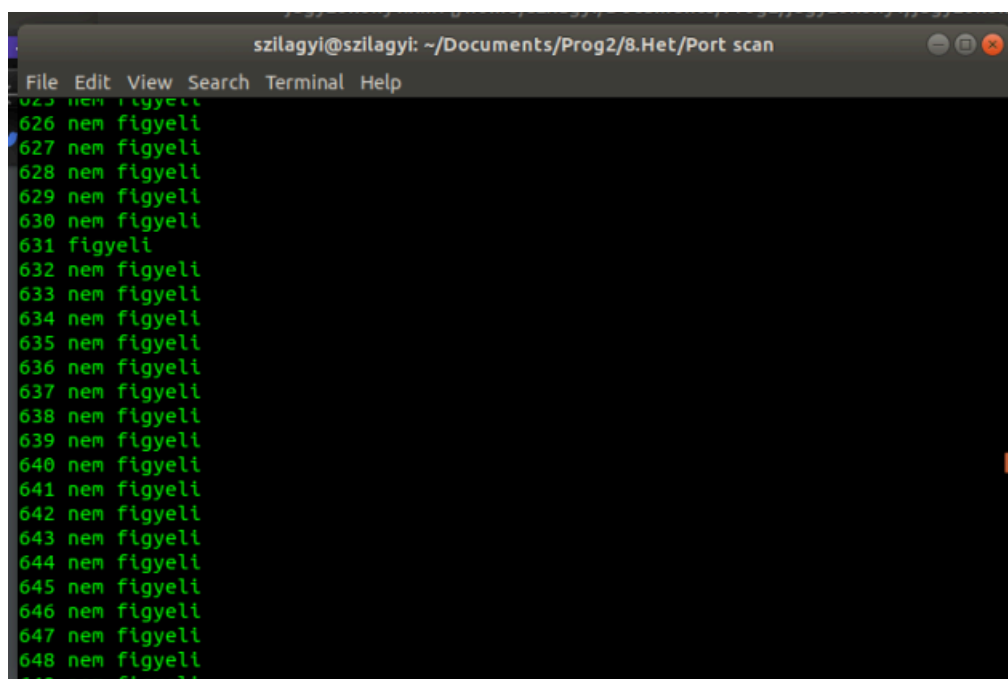
A KapuSzkennel.java segítségével megnézhetjük, hogy gépünk éppen milyen portokat figyel. A kivételkezelés try blokkjában az argumentumként megkapott gép 1024 alatti portjain próbálunk meg TCP kapcsolatot létesíteni. Ha az adott gépen egy szerverfolyamat figyeli az adott portot, akkor ezt ki is iratjuk.

```
public class KapuSzkennel {  
    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();  
            }  
        }  
    }  
}
```

Portszkennelést például akkor használunk, mikor szeretnénk meggy#z#dni arról, hogy egy általunk üzemeltetett szerver nem fed fel túl sokat (tehát nem rendelkezik feleslegesen nyitva lévő portokkal).

Ezen program m#k#dési elve, hogy 0-tól 1024-ig végigmegy az összes porton, és megpróbál egy socketet nyitni a java.net.Socket osztály segítségével. Amennyiben elolvassuk az ehhez az osztályhoz tartozó dokumentációt, azt láthatjuk, hogy amennyiben nem tudunk csatlakozni, úgy SecurityException hibát fog dobni a meghívás.

Ennek köszönhet#en egyszer#en, amennyiben nem tudunk socketet nyitni az adott porton, tudjuk, hogy nem elérhet#.



```

szilagyi@szilagyi: ~/Documents/Prog2/8.Het/Port scan
File Edit View Search Terminal Help
szilagyi@szilagyi:~/Documents/Prog2/8.Het/Port scan$ java KapuSzkenner localhost
| grep -v nem
631 figyel
szilagyi@szilagyi:~/Documents/Prog2/8.Het/Port scan$

```

AOP

Sz#j bele egy átszöv# vonatkozást az els# védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Néhány alap fogalom A csatlakozási pont vagy joint pont egy olyan pont a programban amikor valami történik. Ilyen például amikor egy metódus meghívódik vagy egy kivételt throwolunk, inicializálunk egy objektumot stb. A pointcutnak van egy jobb és bal oldala kettospont választja el a kett#t. Bal oldalon a pointcut neve és paraméterei állnak. A paraméterekben az elérhet# adatokhoz férhetünk hozzá ha lefut a metódus. A jobboldalt maga a pointcut szerepel. Ez általában a call vagy az execution. Ebben írjuk meg, hogy mire utal a pointcut a valódi osztályunkban. Advice-ok lényegében a before, after és around. Tehát, hogy a metódus hívása előtt, után vagy közben # esetleg helyett fusson le. A within paranccsal mondhatjuk meg, hogy melyik osztályból akarjuk használni. A target az a java objektum ami a metódust hívja. Argumentumok azok az értékek amiket a metódusban hívunk.

```

private long egyes = 0;
private long nullas = 0;
pointcut pushback() : execution(public void push_back(char));
before(char ch): pushback() && args(ch){
if(ch == '1'){
egyes++;
} else {
nullas++;
}
}
pointcut main() : execution(public static void main(String[]));
after() : main(){
System.out.println("Egyesek száma: " + egyes);
System.out.println("Nullások száma: " + nullas);
}

```

Vegyük elsonék példának az egyesek és nullások megszámlálását. Ugyebár deklarálunk és inicializálunk 2 változót amibe majd tároljuk a megszámlolt értékeket. Aztán jön a pointcutunk aminek a neve pushback lesz. A jobb oldala maga a point cut. Tehát oda tesszük a pointcutot ahol lefut a fo program push_back(char) függvénye. Aztán mielőtt ez lefut az argumentum karakterjét átadjuk és megnézzük egyes-e vagy nulla. Majd létrehozunk egy új pointcutot a mainre. És megmondjuk, hogy miután lefut a main írjuk ki a megszámlolt egyesek és nullások számát. Most nézzük meg a kiír függvényre való pointcut-unkat:

```

public pointcut meghiv(LZWBInFa.Csomopont n, PrintWriter os)
: call(void LZWBInFa.kiir(LZWBInFa.Csomopont, PrintWriter)) && args(n,os);
after(LZWBInFa.Csomopont n, PrintWriter os) : meghiv(n, os)
{
try{
os=new PrintWriter("preorder.txt");
preOrder(n,os);
os.flush();
}
catch (FileNotFoundException e) {
e.printStackTrace();
}
}

```

```

    }
    depth = 0;
    try{
        os=new PrintWriter("postorder.txt");
        postOrder(n,os);
        os.flush();
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

public void preOrder(LZWBinFa.Csomopont n, PrintWriter p)
{
    if (n != null)
    {
        ++depth;
        for (int i = 0; i < depth; ++i)
            p.print("---");
        p.print(n.getBetu () + "(" + depth + ")\n");
        preOrder (n.getBalNulla (), p);
        preOrder (n.getJobbEgy (), p);
        --depth;
    }
}

public void postOrder(LZWBinFa.Csomopont n, PrintWriter p)
{
    if (n != null)
    {
        ++depth;
        postOrder (n.getBalNulla (), p);
        postOrder (n.getJobbEgy (), p);
        for (int i = 0; i < depth; ++i)
            p.print("---");
        p.print(n.getBetu () + "(" + depth + ")\n");
        --depth;
    }
}

```

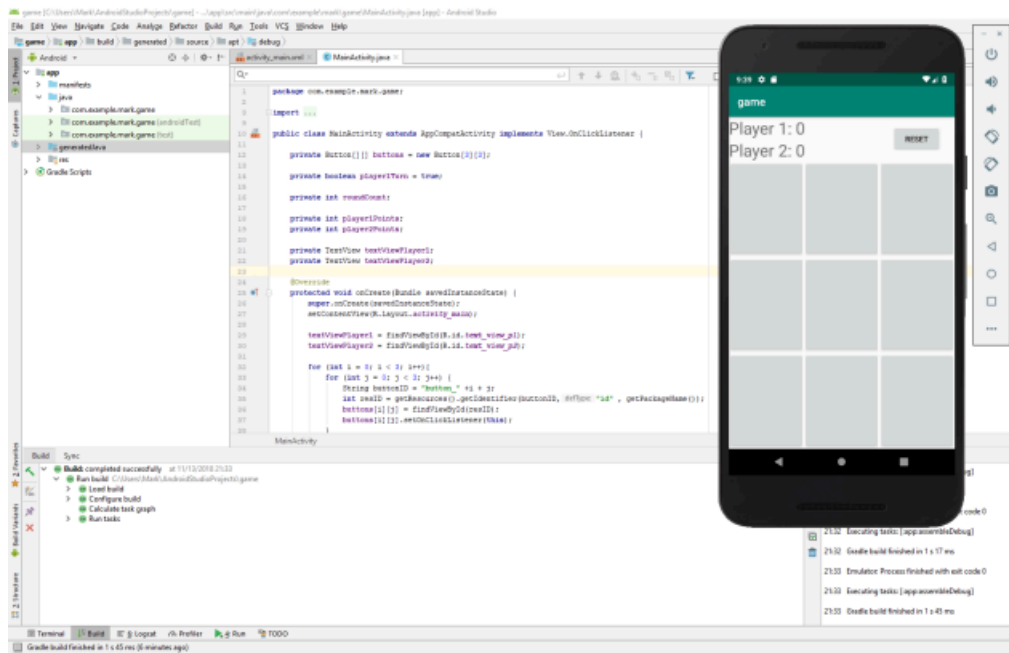
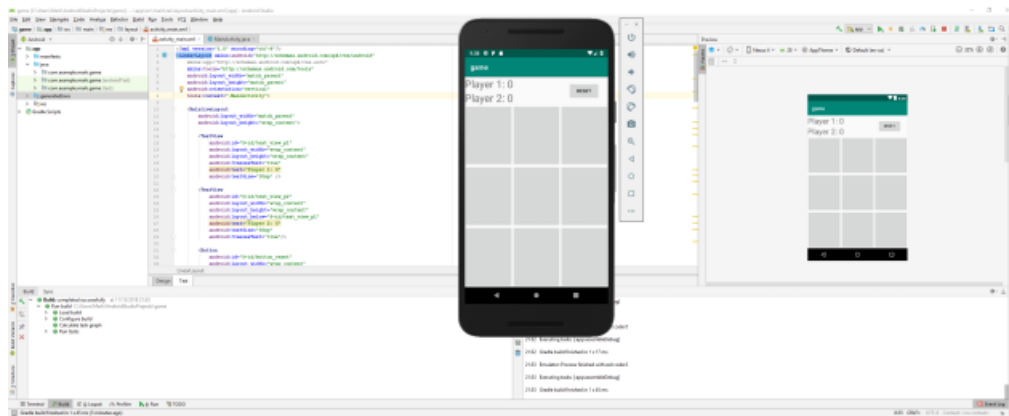
Na itt a pointcutunkat a kiír függvény hívására tesszük. Argumentumként átvesszük a printwriter és a csomópont n-t. Aztán a már megszokott pre és post order bejárásunkat láthatjuk függvényként ez bevprogról már ismeros volt. Ugyebár alapból inorder-ben íratjuk ki. Aztán megmondjuk, hogy minden egyes kiír hívás után hajtódjon végre a másik két kiír függvény is. Fontos a try-catch szerkezet máskülönben errort kapunk, hogy nincs ami elkapja az exceptiont. Emellett nagyon fontos szerepet játszik még az, hogy 2 FileWriter os-t hozok létre. És mind a 2-t flush-ölöm. Ugyebár a flush a bufferben lévő stringet azonnal kiírja.

Android Játék

Írjunk egy egyszer# Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

Egy m#ködo androidos játékot kellett írni, amit én ezt a alábbi youtube tutorial alapján valósítottam meg: <https://www.youtube.com/watch?v=apDL78MFR3o> A játék egy tictactoe játék, ami a játék szabályai alapján m#kodik, számolja a játékosok pontjait, valamint egy reset funkció is bele van építve, ami az eddig összegy#jtött pontokat nullázza.

8. hét - „Helló, Lauda!”



Chapter 11. 9. hét - „Helló, Calvin!”

13. hét Multiparadigmás nyelvek és 14. hét Programozás multiparadigmás nyelveken. Összevonva.

MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel,

https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol Háttérként ezt vetítsük le:

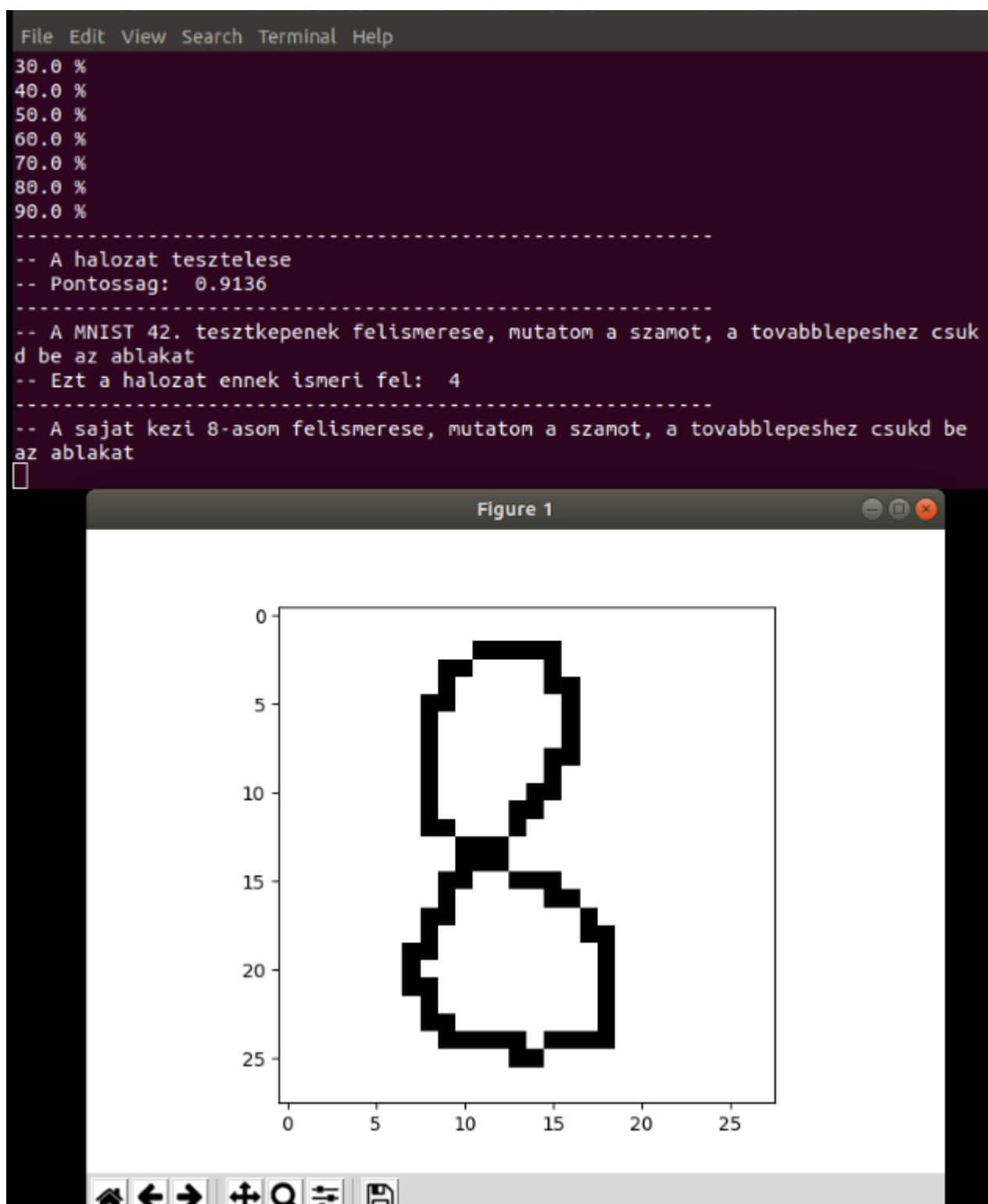
<https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Az MNIST adatállomány kézzel írott számok képeit tartalmazza, pontosabban 60 ezer képet használ a training fázisban, és 10 ezer képet a tesztelés során, ezeknek a képeknek a segítségével tudja megállapítani a felhasználó által inputként megadott 28x28-as képről, hogy az milyen számjegyet takar.

Mindegyik kép 28x28 pixel, a képet átalakítjuk vector formájúra, amely $28 \times 28 = 784$ komponenset tartalmaz. Az adatállomány mindegyik eleméhez egy címke is tartozik, amely egy 0 és 9 közötti számjegy, s a képen látható számot mutatja. Esetünkben a címke egy 10 komponensű „one-hot” vektor. Az 1 számjegyet egyetlen pozíción, az n-ediken tartalmazza, annak megfelelően, hogy melyik számjegy látható a képen, a többi komponens 0. A gépi tanítási elvet követve, adataink egy részét – többnyire a többségét – tanításra, egy további részét modellünk jószágának mérésére használjuk. A bemeneti tenzor 784 db számból áll például egy pont koordinátái a 784 dimenziós térben. Az eredmény pedig 10 db érték 0 és 9 közötti számokat tartalmazza, ami azt jelenti, hogy a rendszer százalékosan fejezi ki, hogy szerinte melyik szám van a képen, például egy írott 6-osra azt mondja, hogy 10%, hogy 8, 20% hogy 9, és 70%, hogy 6-os számot lát. A MNIST 85%-os pontossággal tudja helyesen megállapítani egy kézzel írott számjegyről, hogy az pontosan melyik számjegy.

Ha egy egyedről/tárgyról el kell döntenünk, hogy több különböző egyed/tárgy közül melyikkel milyen valószínűséggel egyezik meg, akkor erre a célra a „softmax” függvény használható, mivel a „softmax” megad egy listát az egyezési valószínűségekre vonatkozóan, ahol az értékek 0 és 1 közöttiek, és az összegük 1, tehát a „softmax” valószínűségeloszlást ad meg, egy x inputhoz kiszámítja az egyes osztályokba tartozás súlyait, azután megadja az osztályokba tartozási valószínűségeket.

A TensorFlow egy nyílt forráskódú, alacsonyabb szintű neurális háló library, ami többek között lehetőséget nyújt neurális hálók összerakására is. A TensorFlow könyvtárban rendelkezésre álló segédanyagok lehetővé teszik színvonalas modellek gyors és egyszerű létrehozását. A TensorFlow rendszerben kifejlesztett számítások változatlanul vagy csekély változtatással végrehajthatók nagyon eltérő hardver eszközökön a mobil telefonoktól és tabletektől kezdve, grafikus kártyákon (GPU) át, sok számítógépből álló elosztott számítógép-rendszerekig. A TensorFlow roppant flexibilis, nagyon széles körű algoritmusok megvalósítására alkalmas, beleértve a deep neural network – sokrétegű neurális háló – alkalmazásait, például a beszédfelismerésben, a számítógépi látásban, megjelenítésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében, és az agykezelésben. TensorFlow-val tenzor transzformációs gráfokat lehet összerakni. A TensorFlow számítás egy irányított gráf írja le. Adatáramlás a gráf élei mentén történik. A TensorFlow gráfban mindegyik csúcs egy m#veletet reprezentálhat és mindegyik csúcshoz lehet nulla vagy több inputja, ugyanígy nulla vagy több outputja. A gráf normál élei mentén áramló értékek tenzorok, tetszőleges dimenziójú vektorok. Egy-egy elem típusát a gráf konstruálásakor specifikálják. Lehetnek a gráfban speciális élek is, amelyek mentén nem történik adatáramlás, hanem kontrol célokat szolgálnak. A használat el#tt a TensorFlow-t importálni kell: `import tensorflow as tf`. A m#velet végzéséhez egy x változó definiálása: `x=tf.placeholder(tf.float32, [None, 784])`. A modell implementálása mindössze egyetlen sor: `y=tf.nn.softmax(tf.matmul(x,W)+b)`.



CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel,

https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol

A CIFAR egy gépi tanulási adatkészlet, ami képek segítségével lehet tanítani, hogy felismerje, hogy milyen képen mi található. 32x32 felbontású képekkel dolgozik. Az alap adatszerkezetében több millió kép található, de lehet használni saját képeket is, és így lehet tanítani.

Elsonek is be kell tanítani a programot a képek felismerésére. Ehez a képekből bináris álmányt kell készíteni, amit a program értelmezni tud. Ezért kell egy olyan algoritmus, ami becsomagolja a képet.

```
from PIL import Image
import numpy as np
im = Image.open('image.png')
im = (np.array(im))
r = im[:, :, 0].flatten()
g = im[:, :, 1].flatten()
b = im[:, :, 2].flatten()
label = [1]
out = np.array(list(label) + list(r) + list(g) + list(b), np.uint8)
out.tofile("out.bin")
```

A betanítás után jöhet a kép felismerése. Ezt az elkészített bináris álmányra kell elvégezni, úgy hogy a példaprogram `cifar10_eval.py` fájlt kicsit módosítani kell, hogy ne számoljon pontosságot, mert a feladat szempontjából teljesen lényegtelen, itt csak az kell, hogy melyik csoportba sorolható bele a kép.

```
.
.
.
# while step < num_iter and not coord.should_stop():
# predictions = sess.run([top_k_op])
predictions = sess.run([top_k_op])
print(sess.run(logits[0]))
classification = sess.run(tf.argmax(logits[0], 0))
cifar10classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]
print(cifar10classes[classification])
true_count += np.sum(predictions)
step += 1
# Compute precision @ 1.
precision = true_count / total_sample_count
# print('%s: precision @ 1 = %.3f' % (datetime.now(), precision))
.
.
.
```

Itt látszik is a módosítás, hogy a pontosság ki van kommentelve, és meg vannak adva a felismeréshez szükséges osztályok. Az ellenőrzés csak egyszer hajtódik végre, majd meg kell keresni a legnagyobb indexel rendelkező értéket, majd meghívásra kerül rá az `sess.run` funkció, így elkezdődik a felismerés és visszatér a megtalált értékkel, utána a `cifar10classes` tömbből kikérdezi a típust, majd kiírja.

Mivel csak egy darab képet ismertettünk fel a programmal ezért még a `cifar10.py` fájlban módosítani kell a `batch_size` értékét 1-re.

Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

A TensorFlow egy szoftverkönyvtár, mely lehetővé teszi gépi tanulási algoritmusok végrehajtását. Ezek a számítások nemcsak számítógéprendszereken, de akár mobiltelefonokon is végrehajthatóak. A TensorFlow számítást egy irányított gráf írja le, melynek élein keresztül adatok (tenzorok) áramlása történik. A gráf csúcsai műveleteket reprezentálnak, a csúcsok inputjainak és outputjainak száma lehet nulla vagy több. A gráfok szerkezetének áttekintését a `TensorBoard` nevű vizualizációs eszközzel érhetjük el. A TensorFlow segítségével tehát képesek vagyunk betanítani Androidos készülékünket különféle objektumok detektálására. A teszteléshez én a https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android projektet használtam. Ez az objektumfelismerő alkalmazás a Google TensorFlow gépi tanulás modelljét használja fel. A hátsó kamera segítségével

valós időben próbálja érzékelni a képen látható tárgyakat. A tárgy nevén kívül egy 2 tizedesjegy# számértéket (százalékot) is kapunk, amely azt érzékelteti, hogy mennyire biztos a felismerésben. Az alkalmazás viszonylag sok nyelvet ismer, magyar nyelv hiányában én a tárgyak angol megfelelőit kaptam. Az informatikai eszközök felismerésével nem volt különösebb problémám. Jó értékeket és megnevezést kaptam többek között egérre, billentyűzetre, ollóra is.

