

---

# Magas szintű programozási nyelvek

## 2. Jegyzőkönyv. Kovács Attila Patrik.

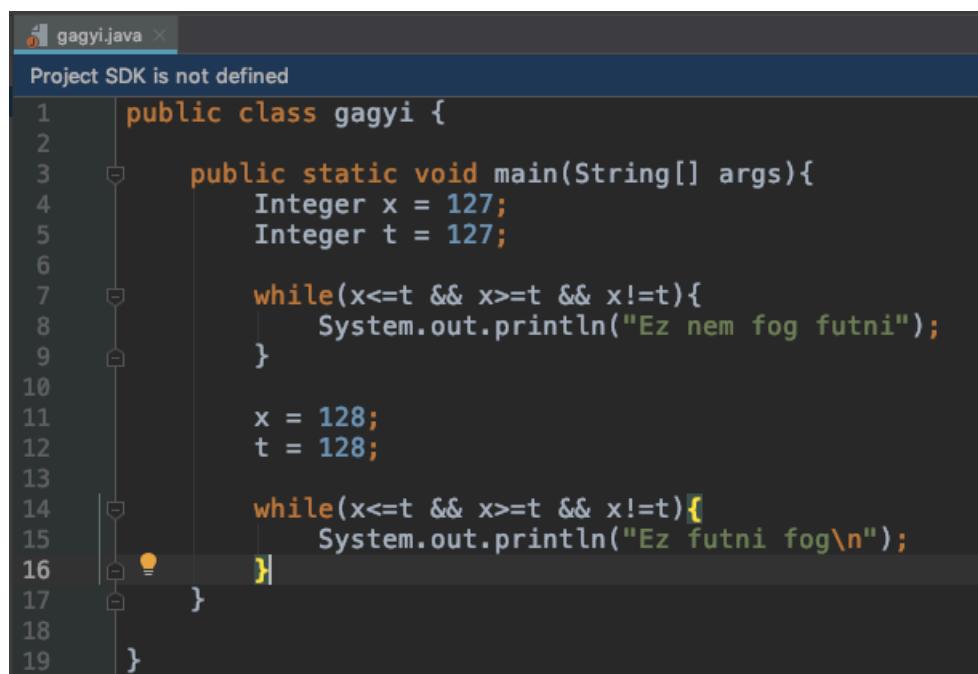
### Table of Contents

1. hét .....	2
Gagyi .....	2
Yoda .....	3
OO szemlélet .....	4
Kódolás from scratch .....	6
2. hét .....	8
Liskov helyettesítés sértése .....	8
Ciklomatikus komplexitás .....	10
Szülő-gyerék .....	11
Anti OO .....	13
3. hét .....	14
Reverse engineering .....	14
Forward engineering .....	15
Egy esettan .....	15
BPMN .....	16
4. hét .....	17
Encoding .....	17
Perceptron osztály .....	19
Full screen .....	21
l334d1c4 <sup>5</sup> .....	23
5. hét .....	25
JDK osztályok .....	25
Változó argumentumszámláló ctor .....	27
Másoló-mozgató szemantika .....	28
Összefoglaló (Másoló-mozgató szemantika) .....	28
6. hét .....	30
C++11 Custom Allocator .....	30
STL map érték szerinti rendezése .....	31
Alternatív Tabella rendezése .....	32
Gengszterek .....	34
7. hét .....	35
FUTURE tevékenység editor .....	35
OOCWC Boost ASIO hálózatkezelése .....	36
OSM térképre rajzolása .....	37
BrainB .....	39
8. hét .....	40
Port scan .....	40
Android Játék .....	42
JUnit teszt .....	46
AOP .....	47
9. hét .....	49
MNIST .....	49
Deep MNIST .....	51
CIFAR-10 .....	56
Android telefonra a TF objektum detektálója .....	57
0. hét .....	65
C++ és Java .....	65
Python .....	66

## 1. het

### Gagyi

Ha -128 és 127 közötti értékeket írunk, akkor a while feltétel hamis lesz, és nem megy bele a ciklusba. A csavar ott van, hogy -128 és 127 (kommentben ott a range -128tól 127ig) értékek között egy előre elkészített poolból kapjuk meg az értéknek megfelelő objektumot, ezért kétszer a 127 nyílván ugyanaz lesz. Viszont ha nem a range közötti értéket adunk, akkor az Integer.java-ban a return new Integer(i) fut le, azaz két eltérő című ÚJ objektumunk lesz, és már is igaz lesz az  $x \neq t$  is. Érdekes még a történetben, hogy hasonlító jelek nél úgy érték szerint hasonlít össze, == vagy != esetén pedig a memóriacímeket nézi.



The screenshot shows a Java code editor window with the file 'gagyi.java' open. The code is as follows:

```
1  public class gagyi {
2
3      public static void main(String[] args){
4          Integer x = 127;
5          Integer t = 127;
6
7          while(x<=t && x>=t && x!=t){
8              System.out.println("Ez nem fog futni");
9          }
10
11         x = 128;
12         t = 128;
13
14         while(x<=t && x>=t && x!=t){
15             System.out.println("Ez futni fog\n");
16         }
17     }
18 }
19 }
```

A red vertical bar highlights the condition `x<=t && x>=t && x!=t` in the second while loop. Above the code, a blue bar displays the message "Project SDK is not defined".

```
1997 ▼    private static class IntegerCache {  
1998        static final int low = -128;  
1999        static final int high;  
2000        static final Integer cache[];  
2001  
2002    static {  
2003        // high value may be configured by property  
2004        int h = 127;  
2005        String integerCacheHighPropValue =  
2006            VM.getSavedProperty("java.lang.Integer.IntegerCache.high");  
2007    if (integerCacheHighPropValue != null) {  
2008        try {  
2009            int i = parseInt(integerCacheHighPropValue);  
2010            i = Math.max(i, 127);  
2011            // Maximum array size is Integer.MAX_VALUE  
2012            h = Math.min(i, Integer.MAX_VALUE - (-low) -1);  
2013    } catch( NumberFormatException nfe ) {  
2014        // If the property cannot be parsed into an int, ignore it.  
2015    }  
2016}  
2017    high = h;  
2018  
2019    cache = new Integer[(high - low) + 1];  
2020    int j = low;  
2021    for(int k = 0; k < cache.length; k++)  
2022        cache[k] = new Integer(j++);  
2023  
2024    // range [-128, 127] must be interned (JLS7 5.1.7)  
2025    assert IntegerCache.high >= 127;  
2026}  
2027  
2028    private IntegerCache() {}  
2029}  
  
1046    @HotSpotIntrinsicCandidate  
1047    public static Integer valueOf(int i) {  
1048        if (i >= IntegerCache.low && i <= IntegerCache.high)  
1049            return IntegerCache.cache[i + (-IntegerCache.low)];  
1050        return new Integer(i);  
1051    }
```

Ha az érték, amit megadunk -128 és 127 közötti, akkor a valueOf nem fog létrehozni egy új objektumot, hanem a cachen belül poololja az Integer objektumot (ugyanazt használja fel). Ha viszont az érték kívül esik a tartományon, akkor a new Integer fut le, ami egy új objektumot hoz létre.

## Yoda

A yoda conditions az összehasonlításnál lép a színrre. Van két változónk, *a* és *b*. Az *a* legyen "valami" a *b* pedig legyen null. Összehasonlításnál ha a *b*-t tesszük előre, akkor "null pointer exception"-t kapunk, mivel nullal nyílván nem hasonlítunk össze. Viszont ha *a*-t tesszük előre, akkor végrehajtja az összehasonlítást.

```
1 public class yoda {  
2  
3     public static void main(String[] args){  
4         String Valami = null;  
5  
6         System.out.println("akarmi".equals(Valami));  
7  
8         System.out.println("-----");  
9  
10        System.out.println((Valami).equals("akarmi"));  
11    }  
12}  
13  
14}
```

```
Kovacs-MacBook-Air:yoda kovacsattila$ java yoda  
false  
-----  
Exception in thread "main" java.lang.NullPointerException  
at yoda.main(yoda.java:10)
```

## OO szemlélet

A mainben kiiratom a next()-et 1-től 5-ig. Ha éppen nincs tárolt érték, akkor befut a generálásba, és csinál két randomot (stored és k), és ebben az esetben a k-t visszaadja. Ha pedig van tárolt érték, akkor csak simán visszaadja(stored).

```
1 import java.text.DecimalFormat;
2
3 public class polar {
4
5     double stored;
6     boolean nostored = true;
7
8     public polar(){
9         nostored = true;
10    }
11
12     public double next(){
13         double bfst, bsec, chelr, k;
14         if(nostored == true) {
15             do {
16                 bfst = 2 * Math.random() - 1;
17                 bsec = 2 * Math.random() - 1;
18                 chelr = bfst * bfst + bsec * bsec;
19             } while (chelr > 1);
20
21             double cont = Math.sqrt((-2 * Math.log(chelr)) / chelr);
22             stored = cont * bsec;
23             nostored = !nostored;
24             k = cont * bfst;
25             return k;
26         }
27         else {
28             nostored = !nostored;
29             return stored;
30         }
31     }
32
33     public static void main(String[] args) {
34         polar newp = new polar();
35         DecimalFormat df = new DecimalFormat("#.#####");
36         for ( int i = 0; i < 5; ++i) {
37             System.out.println(i+1 + ". " + df.format(newp.next()));
38         }
39
40     }
41
42 }
```

- 1. 0,553572
- 2. -0,391501
- 3. 0,550501
- 4. 1,59871
- 5. -1,159561

Egy ehhez nagyon hasonló fgv. dolgozik a Random.java fájlban is(nextGaussian()).

```
583 ▼  public synchronized double nextGaussian() {  
584      // See Knuth, ACP, Section 3.4.1 Algorithm C.  
585 ▼      if (haveNextNextGaussian) {  
586          haveNextNextGaussian = false;  
587          return nextNextGaussian;  
588 ▼      } else {  
589          double v1, v2, s;  
590 ▼          do {  
591              v1 = 2 * nextDouble() - 1; // between -1 and 1  
592              v2 = 2 * nextDouble() - 1; // between -1 and 1  
593              s = v1 * v1 + v2 * v2;  
594          } while (s >= 1 || s == 0);  
595          double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);  
596          nextNextGaussian = v2 * multiplier;  
597          haveNextNextGaussian = true;  
598          return v1 * multiplier;  
599      }  
600  }  
601 }
```

Apróbb különbség a mi kódunk és a JDK között, hogy a JDK StrictMath osztályt használ a mi Math osztályunk helyett, ami annyiban más, hogy pontosabb, mint a Math, viszont lassabb is valamennyivel.

A másik különbség az 583. sorban fedezhető fel: synchronized. Ez szinkronizálta teszi a metódust, ami azt vonja maga után, hogy ha pl. több szál akarja meghívni az objektumra ugyanazt az efféle metódust, akkor egyszerre csak egy lesz végrehajtva, a többi pedig megvárja a végét. És mivel az ilyen metódus hamarabb lefut, mint akármilyen más metódus, amelyet ugyanarra az objektumra hívtak meg, ezért az objektumon belüli változásokat minden másik szál is látni fogja.

## Kódolás from scratch

A program megírásához segítségül vettettem a megadott linket.

A Bailey-Borwein-Plouffe (BBP) algoritmus gondolatmenetét használtam, melynek segítségével hexadecimális kifejtésben tudunk kiírni jegyeket a Pi számban, az előző tagok ismerete nélkül. Adott esetben az 1000001. elemtől számol.

```
1 ▼ import java.io.IOException;
2   import java.util.ArrayList;
3   import java.util.List;
4
5 ▼ public class PiBBP {
6     public static char betuze(double pi)
7     {
8         int kovetkezjegy = (int) (pi*16);
9         char betu = 0;
10        if(kovetkezjegy>=0 && kovetkezjegy<=9 )
11            betu = (char) (kovetkezjegy + '0');
12        else{
13            if(kovetkezjegy==10)
14                betu='A';
15            else{
16                if(kovetkezjegy==11)
17                    betu='B';
18                else{
19                    if(kovetkezjegy==12)
20                        betu='C';
21                else{
22                    if(kovetkezjegy==13)
23                        betu='D';
24                else{
25                    if(kovetkezjegy==14)
26                        betu='E';
27                else{
28                    if(kovetkezjegy==15)
29                        betu='F';
30                }}}}}}
31        return betu;
32    }
33
34    public static double Hexalas(double pi)
35    {
36
37        pi=(pi*16) - (int) (pi*16);
38        return pi;
39    }
40
41 ▼ public static double modulo(double n,double k){
42     long t=1;
43     double r=1;
44     int b=16;
45
46     while(t<=n){
47         t=t*2;
48
49     }
50     t/=2;
51     while(t>0)
```

```
49         }
50             t/=2;
51         while(true)
52     {
53             if(n>=t)
54     {
55                 r=b*r%k;
56                 n=n-t;
57             }
58             t/=2;
59         if(t>=1) {
60             r=Math.pow(r,2)%k;
61         }
62         else
63             break;
64     }
65         return r;
66     }
67     public static void main(String[] args) throws IOException {
68         int d=1000000;
69         double S1 = 0,S4 = 0,S5 = 0,S6 = 0, pi=0;
70         if(d>0)
71     {
72         for (int k = 0; k <= d; k++) {
73             S1=S1+((modulo((d-k),(8*k+1)))/(8*k+1));
74             S4=S4+((modulo((d-k),(8*k+4)))/(8*k+4));
75             S5=S5+((modulo((d-k),(8*k+5)))/(8*k+5));
76             S6=S6+((modulo((d-k),(8*k+6)))/(8*k+6));
77         }
78
79         S1=Math.round(S1);
80         S4=Math.round(S4);
81         S5=Math.round(S5);
82         S6=Math.round(S6);
83
84     }
85
86     pi=4*S1-2*S4-S5-S6;
87
88     String pijegyek = new String();
89     while(pi>0)
90     {
91         pijegyek=pijegyek+betuzes(pi);
92         pi=Hexalas(pi);
93
94     }
95     System.out.println(pijegyek);
96 }
97
98 }
```

```
bc-3d-85-af-c1-46:bbp kovacsattila$ java PiBBP
6C65E5308
```

## 2. hétfő

### Liskov helyettesítés sértése

Kicsit átdolgoztam a pingvin-madarás példát. Én a programban cicákkal és plüsscicákkal játszodozok.

A Liskov elv azt akarja mondani, hogy ha A leszármazottja B-nek, akkor mindenhol gond nélkül használható kell legyen A is, ahol B használható.

Az én példámban a cicák isznak tejet, és "meg akarom itatni" a plüsscicákat is. Ha logikusan gondolkodunk, akkor a plüsscica semmilyen esetben nem fog tejet inni.

Lett egy Java és egy C++ kód is.

```
1 #include <iostream>
2
3 using namespace std;
4
5 ▼ class Cica{
6     public:
7     ▼ virtual void catmilkdrinking(){
8         cout<<"IMDRINKINGMILK"<<endl;
9     };
10    };
11
12 class PlussCica : public Cica{};
13
14 ▼ class ForTest{
15     public:
16     ▼ void vd(Cica &cica){
17         cica.catmilkdrinking();
18     };
19    };
20
21 ▼ int main(){
22     ForTest test;
23     Cica cica;
24     test.vd(cica);
25
26     PlussCica plusscica;
27     test.vd(plusscica);
28 }
```

Kovacs-MacBook-Air:Liskov kov  
Kovacs-MacBook-Air:Liskov kov  
IMDRINKINGMILK /github.com/nbatfai/Sam  
IMDRINKINGMILK vétkezöt, hogy hogyan v

```
1 ▼ class Cica{  
2 ▼   public void catMilkDrinking(){  
3       System.out.println("IMDRINKINGMILK\n");  
4   }  
5 }  
6  
7 ▼ class Test{  
8 ▼   public void vd(Cica cica){  
9       cica.catMilkDrinking();  
10    }  
11 }  
12  
13 class Plusscica extends Cica{}  
14  
15 ▼ class Prog{  
16 ▼   public static void main(String[] args){  
17     Test test = new Test();  
18     Cica cica = new Cica();  
19     test.vd(cica);  
20  
21     Plusscica plusscica = new Plusscica();  
22     test.vd(plusscica);  
23   }  
24 }
```

```
Kovacs-MacBook-Air:Liskov kovacsattila$ java Prog  
IMDRINKINGMILK  
  
IMDRINKINGMILK
```

## Ciklomatikus komplexitás

A ciklomatikus komplexitás egy szoftvermetrika. A metrika egy adott kód alapján határozza meg annak a komplexitását, egy számértékben kifejezve.

A feladat az volt, hogy számoljuk ki valamelyik programunk ciklomatikus komplexitását.

Én egy régebbi RSA töréses progit választottam, és a lizard.ws oldalt használtam a méréshez.

Try Lizard in Your Browser

```
.cpp
```

#include <iostream>  
#include <cstdlib>  
#include <ctime>  
#include <stdlib.h>  
#include <time.h>  
#include <math.h>  
#include <vector>  
  
using namespace std;

Analyse

Ebben a kis ablakban kell először kiválasztani a kód nyelvét, utána becopyzni a kódot a mezőbe.

Code analyzed successfully.

File Type .cpp Token Count 4982 NLOC 293

Function Name	NLOC	Complexity	Token #	Parameter #
lko	4	2	31	
dSzamol	13	3	61	
main	268	7	4867	

Egy ilyen ablakban kapjuk vissza az oldal által számolt adatokat. Szépen lebontva fgv.-kre. Az NLOC érték a sorok számát mutatja a fgv.-ben(csak olyan sorokat, amiben írva is van), a complexity a ciklomatóikus komplexitást, a token pedig a karaktereket számolja.

## Szülő-gyerek

Ennél a feladatnál az volt a dolgunk, hogy megmutassuk, hogy az ősön keresztül csak az ős üzenetei küldhetők. Írtam a Plusscica-ban egy fgv.-t(akarmi()), és amikor a Cica-ra akarom hívni, akkor hibát dob mind Java mind C++ esetén. Fordított esetben, mint a példában is van, gond nélkül működne a dolog, tehát ha Cica-ban van egy fgv.-nk és Plusscica-ra akarom hívni, akkor ebben az esetben fogja tudni, hogy mit akarok tőle.

Itt a C++ kód.

```
5 ▼ class Cica{
6     public:
7 ▼     virtual void catmilkdrinking(){
8         cout<<"IMDRINKINGMILK"<<endl;
9     };
10 };
11
12 ▼ class PlussCica : public Cica{
13     public:
14     virtual void akarmi(){
15         cout<<"Ugysem fogjuk latni"<<endl;
16     };
17 };
18
19 ▼ class ForTest{
20     public:
21     void vd(Cica &cica){
22         cica.catmilkdrinking();
23     };
24
25     public:
26     void akarmivd(Cica &cica){
27         cica.akarmi();
28     };
29 };
30
31 ▼ int main(){
32     ForTest test;
33     Cica cica;
34     test.vd(cica);
35     test.akarmivd(cica);
36
37     PlussCica plusscica;
38     test.vd(plusscica);
39     test.akarmivd(plusscica);
40 }
```

```
Kovacs-MacBook-Air:szulegyerek kovacsattila$ g++ cica.cpp -o cica
cica.cpp:27:14: error: no member named 'akarmi' in 'Cica'
        test.akarmivd(cica);
                  ^~~~~~
```

Itt pedig a Java.

```
1 ▼ class Cica{
2 ▼   public void catMilkDrinking(){
3     System.out.println("IMDRINKINGMILK\n")
4   }
5 }
6
7 ▼ class Test{
8 ▼   public void vd(Cica cica){
9     cica.catMilkDrinking();
10  }
11
12 ▼   public void valamivd(Cica cica){
13     cica.valami();
14   }
15 }
16
17 ▼ class Plusscica extends Cica{
18 ▼   public void valami(){
19     System.out.println("Ugysem irja ki");
20   }
21 }
22
23 ▼ class Prog{
24 ▼   public static void main(String[] args){
25     Test test = new Test();
26     Cica cica = new Cica();
27     test.vd(cica);
28     test.valamivd(cica);
29
30     Plusscica plusscica = new Plusscica();
31     test.vd(plusscica);
32     test.valamivd(plusscica);
33   }
34 }
```

```
Kovacs-MacBook-Air:Szülőgyerek kovacsattila$ javac Prog.java
Prog.java:13: error: cannot find symbol
      cica.valami();
           ^
symbol:   method valami()
location: variable cica of type Cica
1 error
```

## Anti OO

A feladat az volt, hogy hasonlítsuk össze a futási időket a BBP algoritmusos progi esetén.

0. pozíciótól számított  $10^6$ ,  $10^7$ ,  $10^8$  darab jegyét határozzuk meg C, C++, Java és C# nyeleken.

Az eredményeket egy táblázatba foglaltam. Később megkértem a tutoromat, hogy futtassa ő is, hogy összehasonlíthassam a futási időket.

Én gépem: 1,8 GHz-es, kétfogas Intel Core i5 processzor (Turbo Boost gyorsítással 2,9 GHz) 3 MB-os mezosztott L3 gyorsítótárral, 128 GB-os SSD-tároló és 8GB RAM.

Tutorom gépe: 2,5 GHz-es, négyfogas Intel Core i5 processzor (Turbo Boost gyorsítással 3,5 GHz) 6 MB-os mezosztott L3 gyorsítótárral, 500 GB-os SSD-tároló és 8GB RAM.

Én eredményem:

	6	7	8
Java	2.39	25.99	323.45
C	2.59	31.96	354.25
C#	2.51	28.84	342.77
C++	2.67	32.15	359.38

Tutorom eredményei:

	6	7	8
Java	1.45	17.94	209.56
C	1.70	20.18	236.50
C#	1.62	19.99	227.45
C++	1.75	21.08	241.56

Leggyorsabbnak a Java bizonyult a tesztek során, elég nagy ugrásnyi különbséggel( $10^8$  esetén 12mp) követi a C#, aztán C és végül a C++.

## 3. het

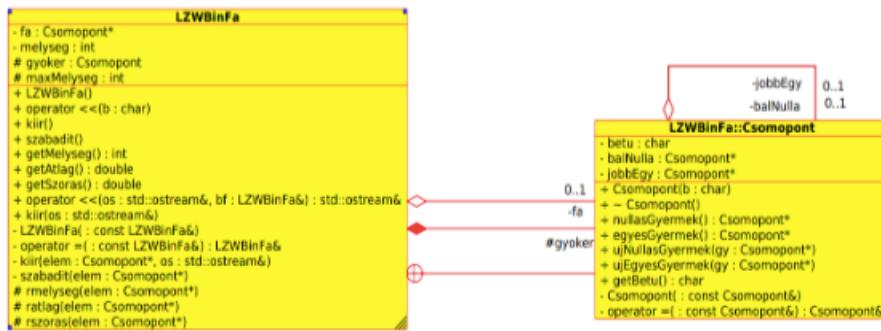
### Reverse engineering

A feladat az volt, hogy rajzoljunk UML osztálydiagramot a tavalyi binfás progihoz.

A feladat során a kódból generáljuk a diagramot. Én Umbrello-t használtam hozzá.

A lényeg itt az, hogy előre dolgozunk, és a már kész kódból készítjük a diagramot.

Ez abban az esetben hasznos mondjuk, ha mások számára akarjuk egyszerübbé tenni a kódunk megértését.



Látszik, hogy nincs csomópont gyökér elem a binfa nélkül, ezt a kompozíciós kapcsolat is mutatja. A két osztály közötti aggregációs kapcsolat pedig az első gyerek elemeket jelöli a képen.

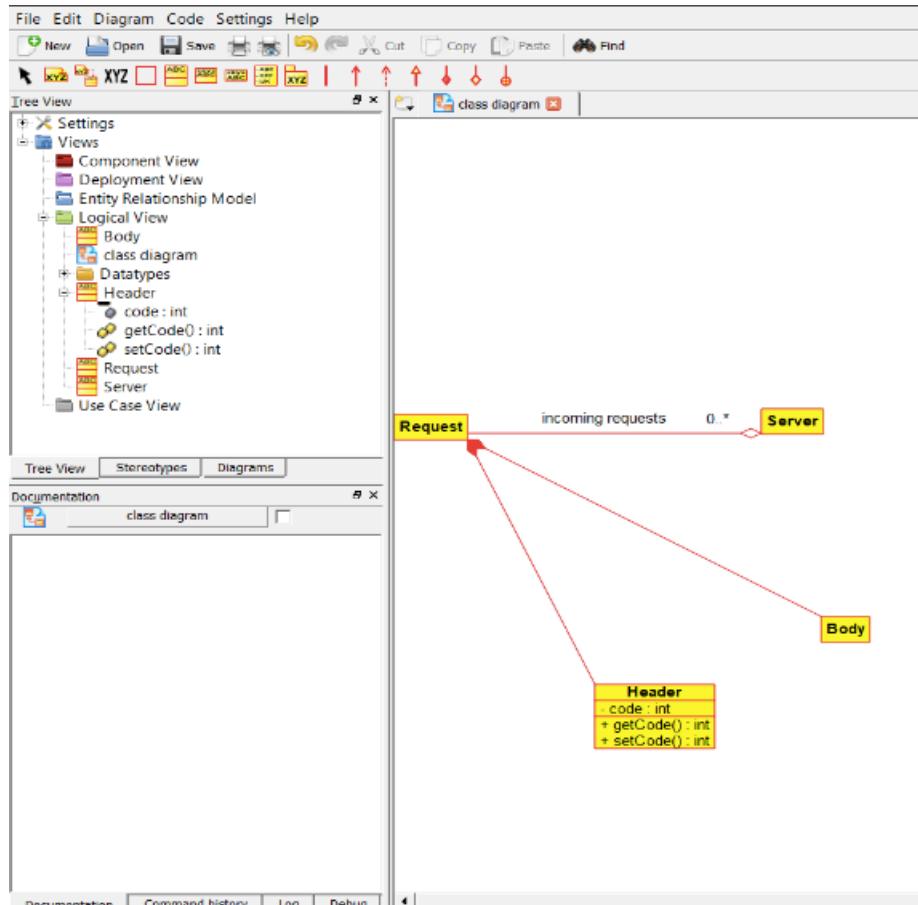
Ahol az asszociációs kapcsolat vissza fordul saját osztályába, ott mutatja, hogy van két aggregáció fajtájú asszociációink is a programban.

A 0..1 fa kapcsolat azt jelöli, hogy pontosan két változat lehet a bevitt paraméterektől függően, mikor nincs gyökér elemünk és mikor van.

## Forward engineering

A feladat az volt, hogy az előző feladattal ellentétben, itt az UML osztálydiagramból csinálunk forrást. Ennek talán egy kicsit több értelme is van.

Itt is Umbrello-t használtam. Először megrajzoltam az egyszerű kis diagramot, aztán a program segítségével kódöt generáltattam belőle.



A diagramunkkal meghatározunk egy szerkezetet a programunknak. Nyílván nem egy kész programot rakunk ezzel össze, de elég hasznos tud lenni olyan esetekben, ha megvan az UML osztálydiagramunk.

A függvények törzse üres marad, nem is tudná szegény Umbrello mivel feltölteni, de kezdetnek elég jó kis induló forrást ad.

## Egy esettan

Az elméletes résznél sok mindenről olvasunk, a könyv írója próbálja részletesen elmagyarázni a modelllezéses, UML-es dolgokat. Szó esik különböző típusú asszociációkról (1-1, több-1, 1-több), kompozícióról, aggregációról és a programnyelvben használt adattípusokról.

Később szó van olyanokról, mint a kódgenerálás, azaz Forward Engineering vagy kódvisszafejtés, azaz Reverse Engineering (ezekről bővebben fentebb félékkel foglalkozunk).

A feladat az volt, hogy létre kell hozni egy olyan alkalmazást, amely a számítógépek és alkatrészek eladásával foglalkozó cégen belül úgymond nyílvántartja az éppen meglévő alkatrészeket és konfigurációkat. A megrendelő sok különböző követelmény szabott meg a program működésével és használatosságával kapcsolatban.

A fő része egy keretrendszer osztálykönyvtár formában. A termékek lehetnek merevlemezek, kijelzők és konfigurációk (a konfigurációk plusz összetett elemek).

Ennek a keretrendszernek az osztályait a ProductInventoryLib osztálykönyvtárba fogjuk össze.

Van egy átfogó jellegű Product osztályunk, amelyben az összes termékre jellemző tulajdonságok gyűjtjük össze. Ebben az osztályban 3 változó található: a név, az ár és a beszerzésének dátuma(ezek protectededek), ezeket getterrel érjük el. Metódusok vannak olyanok, ami pl. a termék "régiségét" mutatja meg, vagy az árat számolja ki. Tartalmaz még egy print metódust is, ami kiírja a kívánt termék adatait.

A Product osztáyból pl. CompositeProduct, Display és HardDisk alosztályok is leszármaztathatóak, illetve a CompositeProduct-ból leszármaztatható a ComputerComfiguration alosztály is.

A dátumot adó fgv. a GetDateOfAcquisition(), a nevet a GetName() adja vissza, míg az árat a GetInitialPrice(). Az ár olyan fgv., amelyet minden származtatott osztály felül fog írni, mivel mindenhol másképp kell működjön.

A fájlok itt megtalálhatóak: <https://github.com/kovacsat2000/esettanfiles>

Futás közben:

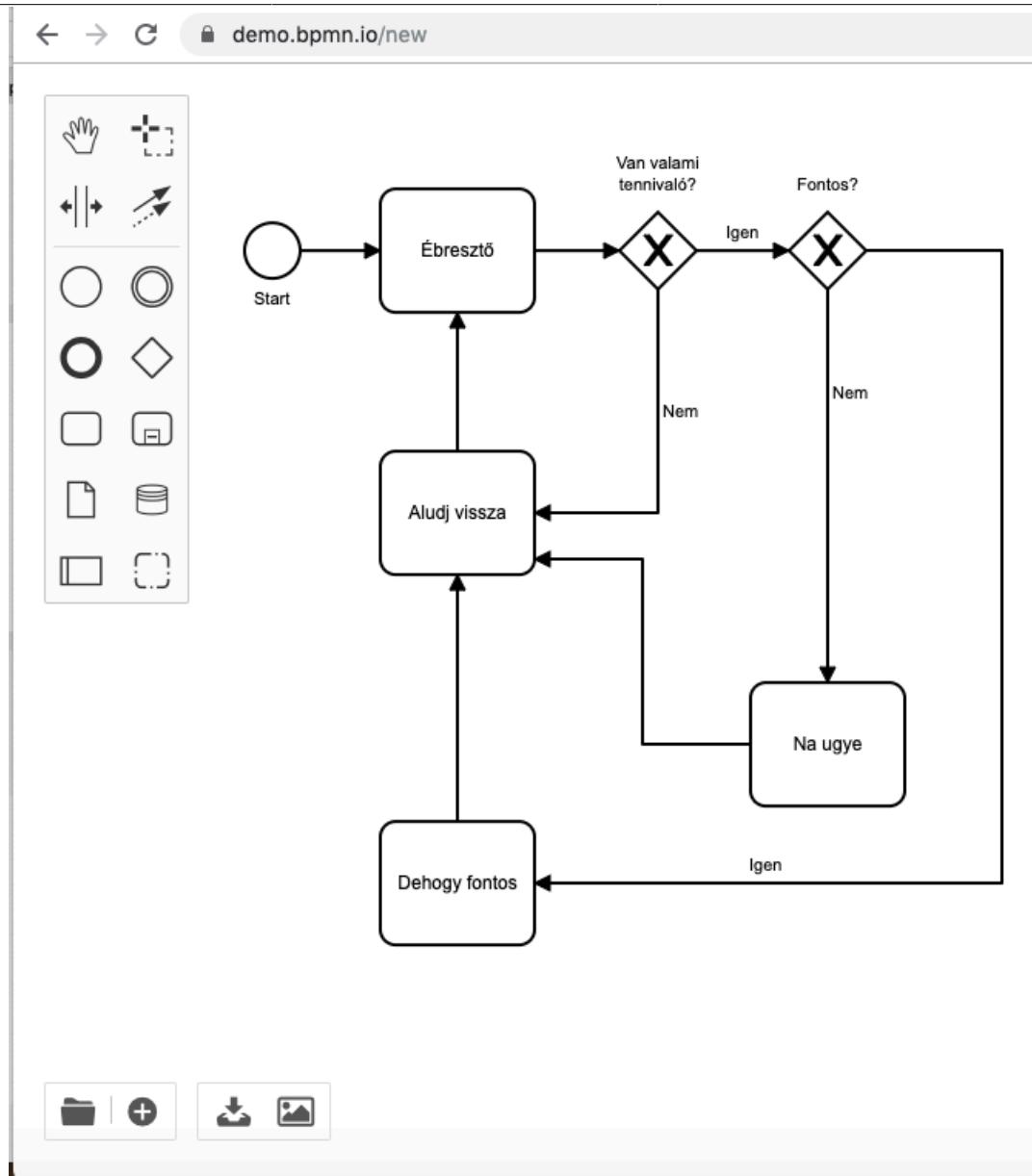
```
Egy esettan — esettan — 80x24
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20060930, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...
Az elmeletes résznl sok mindenröl olvasunk, a könyv frója próbálja részletesen elmagyarázni a
modellezéses UML-es dolgokat. Szó csik különbözö típusú asszociációkról (1-1, több-1, 1-több).
Test2: loading inventory from a file (computerproducts.txt), printing it
en writing it to a file (computerproducts_out.txt).
End of reading product items. The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20060930, Age: 6581, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4756, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4756, Current price: 70000
Items:
0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20060930, Age: 6581, Current price: 24000, InchWidth: 12, InchHeight: 13
1. Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4756, Current price: 28000, SpeedRPM: 7000
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20060930, Age: 5335, Current price: 20000, SpeedRPM: 7000
Cor
Section 3.4: BPMN
The content of the inventory has been written to computerproducts_out.txt
A feladat az volt, hogy rajzolunk le egy tevékenységet BPMN-ben.
Done. Egy egyszerű kis tevékenységet rajzoltam.
```

## BPMN

A feladat az volt, hogy rajzolunk le egy tevékenységet BPMN-ben.

Egy egyszerű kis tevékenységet rajzoltam.

A lényege az, hogy mindegy mi van, akkor is az alvást választjuk.



- 0.) Start
- 1.) Ébresztő → 2.
- 2.) Van valami tennivaló? a)igen → 4. b)nem → 3.
- 3.) Aludj vissza → 1.
- 4.) Fontos? a)igen → 6. b)nem → 5.
- 5.) Na ugye → 3.
- 6.) Dehogy fontos → 3.

## 4. hétfő

### Encoding

A feladat az volt, hogy fordítsuk le és futtassuk a Javat tanítók könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket!

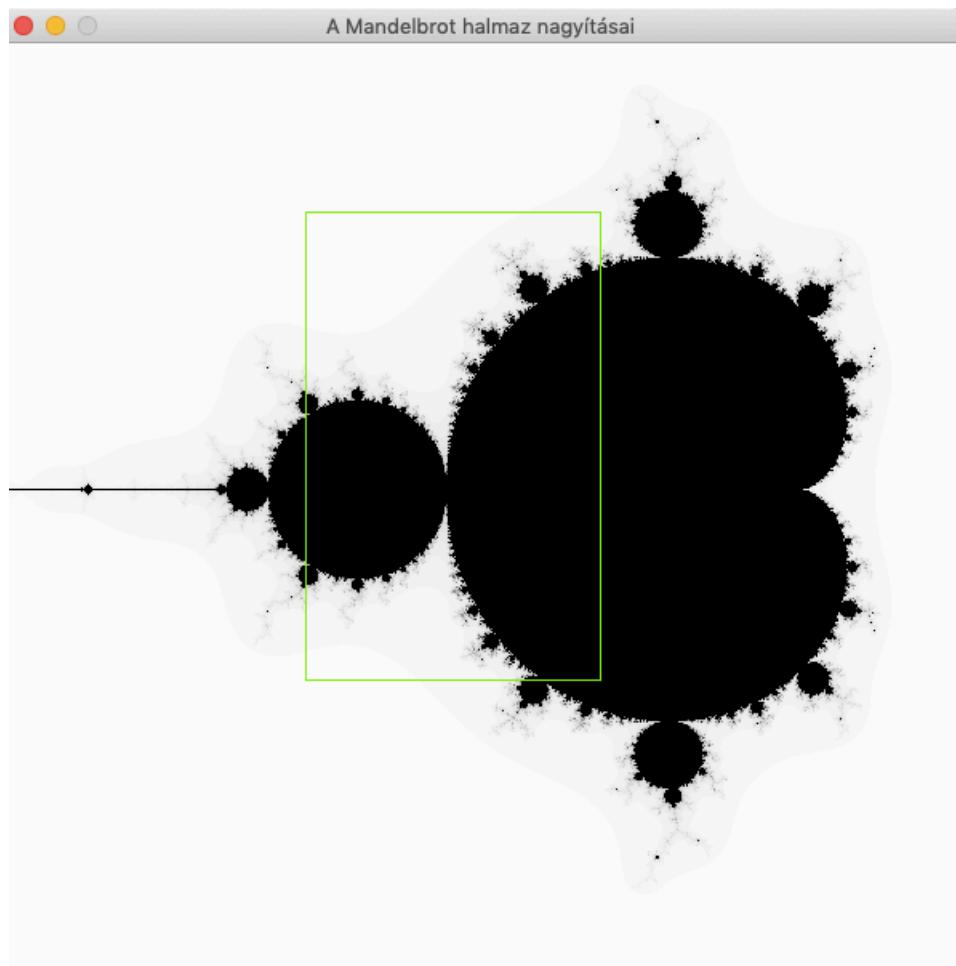
A feladat során a karakterkészletekkel kellett szórakozni... Ha anélkül proóbálon futtatni, hogy a -encoding kapcsolót használnám a javac-nál, akkor laza 100 errorr dob.

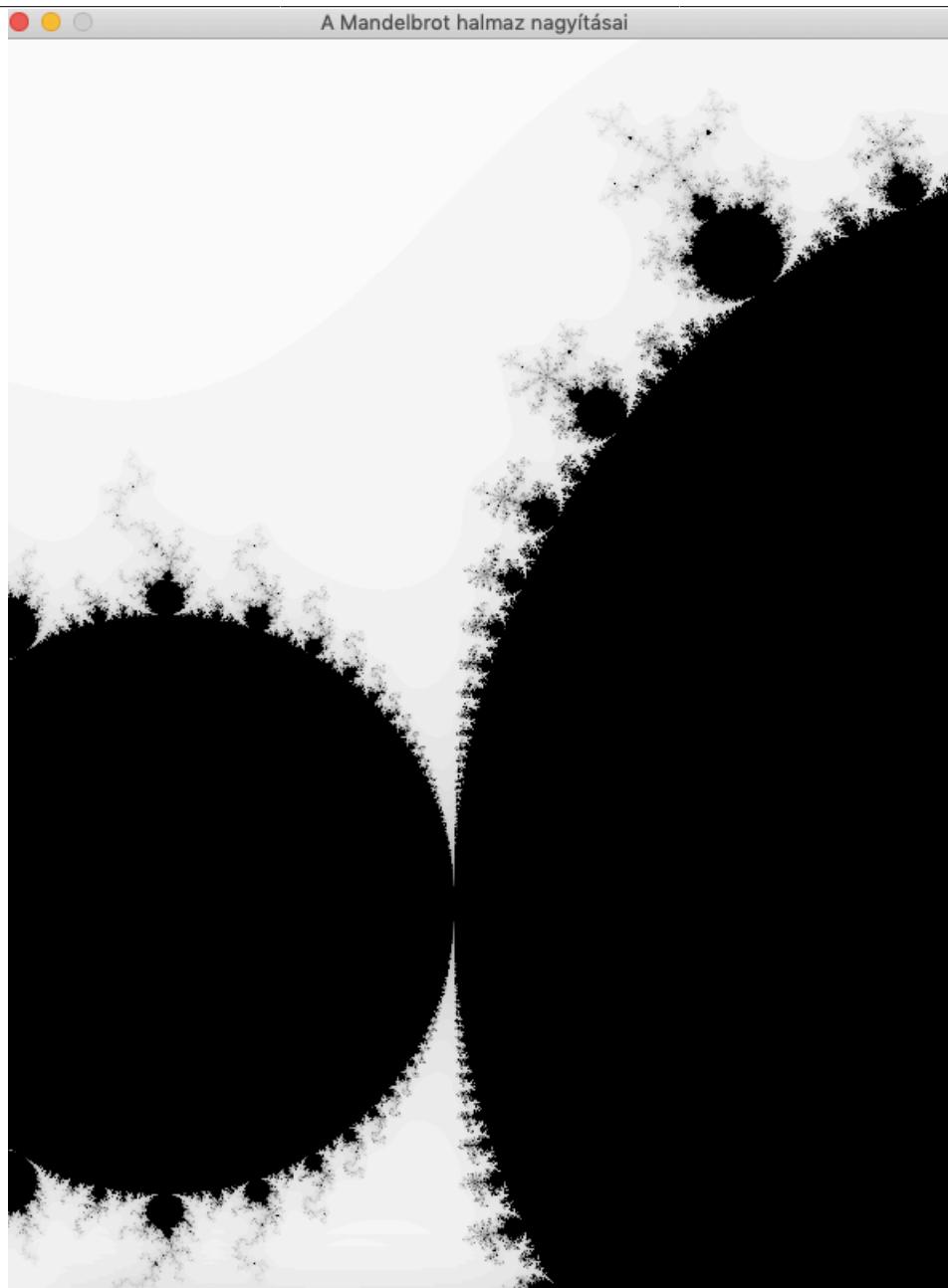
Varázsolgattam a kódolásokkal, kipróbáltam többet is. Sokadjára végül két kódolással is lefordult. Egyik a Latin 2 volt, azaz ISO-8859-2, a másik pedig a windows-1252.

Különösen érdekesnek találtam a feladatot, mert már 9. osztály óta azt hallgatom a progos tanáramtól, hogy minden vagy angolul vagy csúnyán ékezetek nélkül.

```
Kovacs-MacBook-Air:Encoding kovacsattila$ javac -encoding "ISO-8859-2" HalmazNagyító.java
Kovacs-MacBook-Air:Encoding kovacsattila$ javac -encoding "windows-1252" HalmazNagyító.java
```

A progi futás közben így néz ki:





## Perceptron osztály

A feladat az volt, hogy dolgozzuk be egy külön projektbe a Perceptron osztályt!

A pdf-ben lévő link alapján dolgoztam.

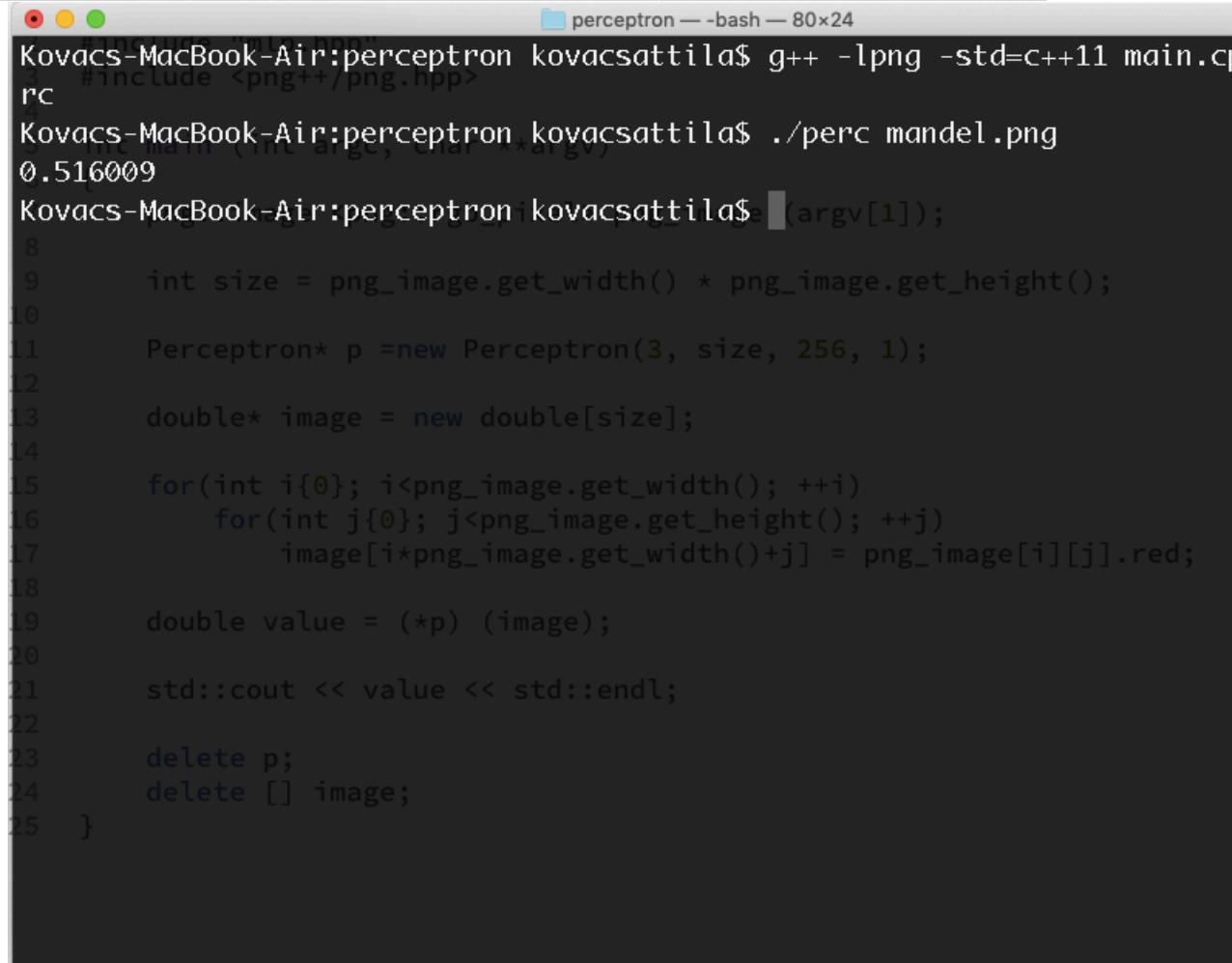
Az mlp.hpp file-ban van maga az osztály.

Ha fordítjuk a programot, utána futtatásnál egy képet kér .png-ben. Én a mandeles képet adtam neki.

És eztán visszaad egy 0 és 1 közötti számot, amivel jellemzi ezt a képet.

A második féléves prog1-en volt egy ilyen feladat, ott kellett dolgozgatni a Perceptron osztállyal.

```
1 ▼ #include <iostream>
2 #include "mlp.hpp"
3 #include <png++/png.hpp>
4
5 int main (int argc, char **argv)
6 ▼ {
7     png::image<png::rgb_pixel> png_image (argv[1]);
8
9     int size = png_image.get_width() * png_image.get_height();
10
11    Perceptron* p =new Perceptron(3, size, 256, 1);
12
13    double* image = new double[size];
14
15    for(int i{0}; i<png_image.get_width(); ++i)
16        for(int j{0}; j<png_image.get_height(); ++j)
17            image[i*png_image.get_width()+j] = png_image[i][j].red;
18
19    double value = (*p) (image);
20
21    std::cout << value << std::endl;
22
23    delete p;
24    delete [] image;
25 }
```



A screenshot of a terminal window titled "perceptron — bash — 80x24". The window shows the following text:

```
Kovacs-MacBook-Air:perceptron kovacsattila$ g++ -lpng -std=c++11 main.cpp
rc #include <png++/png.hpp>
Kovacs-MacBook-Air:perceptron kovacsattila$ ./perc mandel.png
0.516009
Kovacs-MacBook-Air:perceptron kovacsattila$ (argv[1]);
8
9     int size = png_image.get_width() * png_image.get_height();
0
1     Perceptron* p =new Perceptron(3, size, 256, 1);
2
3     double* image = new double[size];
4
5     for(int i{0}; i<png_image.get_width(); ++i)
6         for(int j{0}; j<png_image.get_height(); ++j)
7             image[i*png_image.get_width()+j] = png_image[i][j].red;
8
9     double value = (*p) (image);
0
1     std::cout << value << std::endl;
2
3     delete p;
4     delete [] image;
5 }
```

A program a kép piros pontjaival dolgozik, ahogy a 17. sorban is látjuk, a tömbbe a red kódokat tölti fel.

A sigmoid fgv. számolja a kapott számunkat. A () operátort az mlp-ben írtuk felül. És a 19. sorban hívjuk.

## Full screen

A feladat az volt, hogy készítsünk egy tetszőleges teljesképernyős Java programot.

Én egy ToDoList-es projekt kódjában írogattam ezt-azt át, és "fullscreen"-essé tettek a progit.

# XYZ kft. Tevékenységi naplója

Felhasználónév: |

Press ESC to exit full-sc

Bejelentkezés

Fix méretű volt az ablak, azzal kellett kicsit szívözni, kisebb-nagyobb javítatás után összeraktam a dolgokat.

Ha a fix méretet rajta hagyom a bizonyos elemeken, akkor nagyításnál elcsúszik az egész összkép, ezért ezeket mind át kellett kódolni, mivel a program alapvetően fix méretűre volt tervezve.

A teljes képernyőt a

```
primaryStage.setResizable(true);  
primaryStage.setFullScreen(true);
```

parancsokkal értem el.

Maga az alkalmazás egyébként egy olyan tevékenységnapló szerű dolog, amit a projektben részt vevő csapattársaimmal olyan emberek számára tervezünk, akiknek a napi tevékenységeik fejbentartása, követése, időzítése és rendezgetése komoly problémákat okoz.

A belépésnél egy login ablak fogad minket, hogy be tudjunk lépni a saját tevékenységeink közé. Itt pedig majd tudunk tevékenységeket hozzáadni a napjainkhoz naptáras lebontásban, címkézni tudjuk a tevékenységeket pl. fontosság vagy súrgősség szempontok alapján, állíthatunk határidőt hozzájuk, és hasonló hasznos dolgok.

## I334d1c4<sup>5</sup>

A feladat az volt, hogy írunk olyan OO Java vagy C++ osztályt, amely leet cipherként működik.

Igazán érdekes feladat volt, nekem mindig is tetszettek az ilyen és ehhez hasonló feladatok.

A progim a leet kódolás szabályai szerint kódolja a bekért szöveget.

Kezdetben ez a nyelv hackerek nyelve volt a neten. Például az ilyen nyelven írt oldalak nehezen megtalálhatóak, ha nem tudjuk pontosan mit is kell keresni.

A progim úgy működik, hogy a bekért szöveget végignézi, és a rework fgv. a switchel kicserélgeti a betűket a leet betűkre. Utáni simán kiírogatja már a "leet"-es betűket.

A leet szótárban egy betűhöz több leet-es betű is lehetne, írtam olyan progit is, ami random választ az egy karakterhez tartozó leet-es betük közül egyet, és azzal cseréli a "kódolandó" szövegen a betűt. Azért ezt tettem be, mert ez elegánsabbnak tűnt.

```
1 import java.util.Scanner;
2 import java.util.Vector;
3
4
5 ▼ public class Leet {
6
7     public static Vector<String> rwrkng(String word, Vector<String> wordvector)
8     {
9
10        word=word.toUpperCase();
11        for(int i=0;i<word.length();i++)
12        {
13            switch (word.charAt(i)) {
14                case 'A' : wordvector.add("4");
15                break;
16                case 'B' : wordvector.add("|3");
17                break;
18                case 'C' : wordvector.add("(");
19                break;
20                case 'D' : wordvector.add("|)");
21                break;
22                case 'E' : wordvector.add("3");
23                break;
24                case 'F' : wordvector.add("|=");
25                break;
26                case 'G' : wordvector.add("6");
27                break;
28                case 'H' : wordvector.add("|-|");
29                break;
30                case 'I' : wordvector.add("|");
31                break;
32                case 'J' : wordvector.add(".]");
33                break;
34                case 'K' : wordvector.add("|<");
35                break;
36                case 'L' : wordvector.add("1");
37                break;
38                case 'M' : wordvector.add("|Y|");
39                break;
40                case 'N' : wordvector.add("N");
41                break;
42            }
43
44            wordvector=rwrkng(word,wordvector);
45
46            for(int i=0;i<wordvector.size();i++)
47                System.out.print(wordvector.get(i));
48
49            System.out.println("");
50        }
51    }
52}
```

```
Kovacs-MacBook-Air:1334d1c4 kovacsattila$ javac Leet.java
Kovacs-MacBook-Air:1334d1c4 kovacsattila$ java Leet
Add meg a kódolni kívánt szöveget:
soli Deo gloria!
501|1)30 610|2|4
Kovacs-MacBook-Air:1334d1c4 kovacsattila$
```

## 5. hét

### JDK osztályok

A feladat az volt, hogy írunk olyan Boost C++ programot, amely kilistázza a JDK összes osztályát, ráengedve az src-re.

A progi nem okozott nagyobb nehézségek, mert prog1-en is volt a fénykardnál olyan feladat, amiben ki kellett listázni a neveket és a jegyeket a future-ból. Már ahoz is kellett a boost, szóval már külön telepítenem sem kellett.

A számlálós részének a lényege az, hogy ha talál egy .java végű fájlt, akkor növeli egyel a változó értékét, és a végén simán kiiratom.

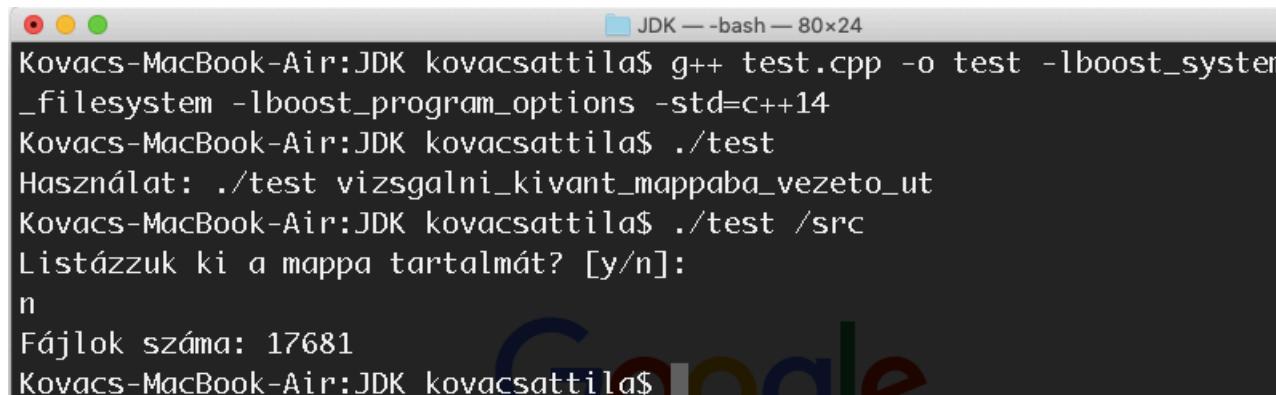
A kód:

```
void bejarTeljes(path path){  
    directory_iterator it{path},eod;  
    BOOST_FOREACH(filesystem::path const& p , make_pair(it, eod)){  
        if (is_regular_file(p) && extension(p.string()) == ".java")  
            cout << p << endl;  
            szamol++;  
        }  
        else if (is_directory(p)) bejarTeljes(p);  
    }  
}  
  
void bejar(path path){  
    directory_iterator it{path},eod;  
    BOOST_FOREACH(filesystem::path const& p , make_pair(it, eod)){  
        if (is_regular_file(p) && extension(p.string()) == ".java")  
            szamol++;  
        }  
        else if (is_directory(p)) bejar(p);  
    }  
}  
  
string a = "";  
cout << "Listázzuk ki a mappa tartalmát? [y/n]: " << endl;  
cin >> a;  
if (a == "y"){  
    Feldolgoz* obj = new Feldolgoz(argv[1]);  
    obj->bejarTeljes(obj->getUt());  
    cout << "Fájlok száma: " <<obj->getSzamol() << endl;  
} else if(a == "n") {  
    Feldolgoz* obj = new Feldolgoz(argv[1]);  
    obj->bejar(obj->getUt());  
    cout << "Fájlok száma: " <<obj->getSzamol() << endl;  
}
```

Azért van két függvény is, mert külön szerettem volna venni, hogy ki szeretnénk-e listázni az összes fájlt, vagy elég csak a mappában lévő fájlok számát.

Aztán létrehozok egy Feldolgoz típusú objektumot és arra engedem rá a fgv-keket egy if-en belül.

Lássuk, hogy működik:



```
JDK — bash — 80x24  
Kovacs-MacBook-Air:JDK kovacsattila$ g++ test.cpp -o test -lboost_system  
_filesystem -lboost_program_options -std=c++14  
Kovacs-MacBook-Air:JDK kovacsattila$ ./test  
Használat: ./test vizsgalni_kivant_mappaba_vezeto_ut  
Kovacs-MacBook-Air:JDK kovacsattila$ ./test /src  
Listázzuk ki a mappa tartalmát? [y/n]:  
n  
Fájlok száma: 17681  
Kovacs-MacBook-Air:JDK kovacsattila$
```

Mint feljebb is írtam, és ezen a képen látszik is, van egy kérdés a lényegi futás előtt, hogy szeretnénk-e a listázott változatot, vagy sem.

Ha y-t írunk, akkor jön a teljes, ha n-t, akkor csak egy számot kapunk.

## Változó argumentumszámú ctor

A feladat az volt, hogy készítsünk olyan példát, amely egy képet tesz a Perceptronos projekt bementére, és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza.

Tehát az mlp.hpp-ben lévő () operátor két argumentumot várt. Lentebb ott a forrás.

Két double típusú tömb a bemenete (double image[], és double image2[]]).

Tehát először létrehozza a második képet, amit majd vissza fogunk adni.

És a továbbiakban belemásolom az eredeti képet, tömbként kezelve a második képhez.

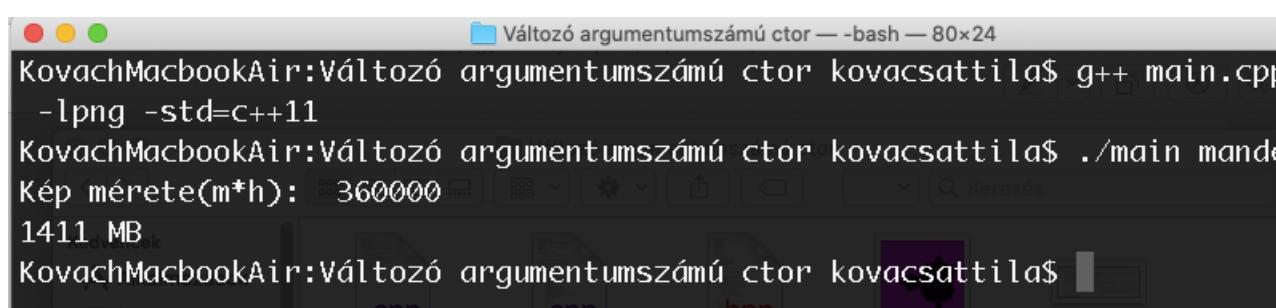
Külön érdekesség, hogy az RGB-ből a green-eket nem teszem bele, így egy kis színcsavar is lesz a képen, tehát nem ugyanolyan lesz a kép. Lentebb látszik.

A main-ben pedig a ()-t két argumentummal hívom. És alatta pedig a write-al belerakom a png-be az új, AZ EREDETIVEL AZONOS MÉRETŰ képet.

```
png::image<png::rgb_pixel>* operator()(double image[],double image2[])
{
    units[0] = image;
    png::image<png::rgb_pixel>* png_image2 = new png::image<png::rgb_pixel>((pow(n_units[0],0.5)),(pow(n_units[0],0.5)));
    for (int k = 0; k < n_units[0];k++)
        image2[k] = units[0][k];
    for (int i{0}; i < int(pow(n_units[0],0.5)); i++)
        for (int j{0}; j < int(pow(n_units[0],0.5)); j++)
    {
        (*png_image2)[i][j].red = image2[i*(int)(pow(n_units[0],0.5)),j].red;
        (*png_image2)[i][j].green = image2[i*(int)(pow(n_units[0],0.5)),j].green;
        (*png_image2)[i][j].blue = image2[i*(int)(pow(n_units[0],0.5)),j].blue;
    }
    return png_image2;
}

png::image<png::rgb_pixel>* value = (*p)(image,image2);
value->write("created.png");
```

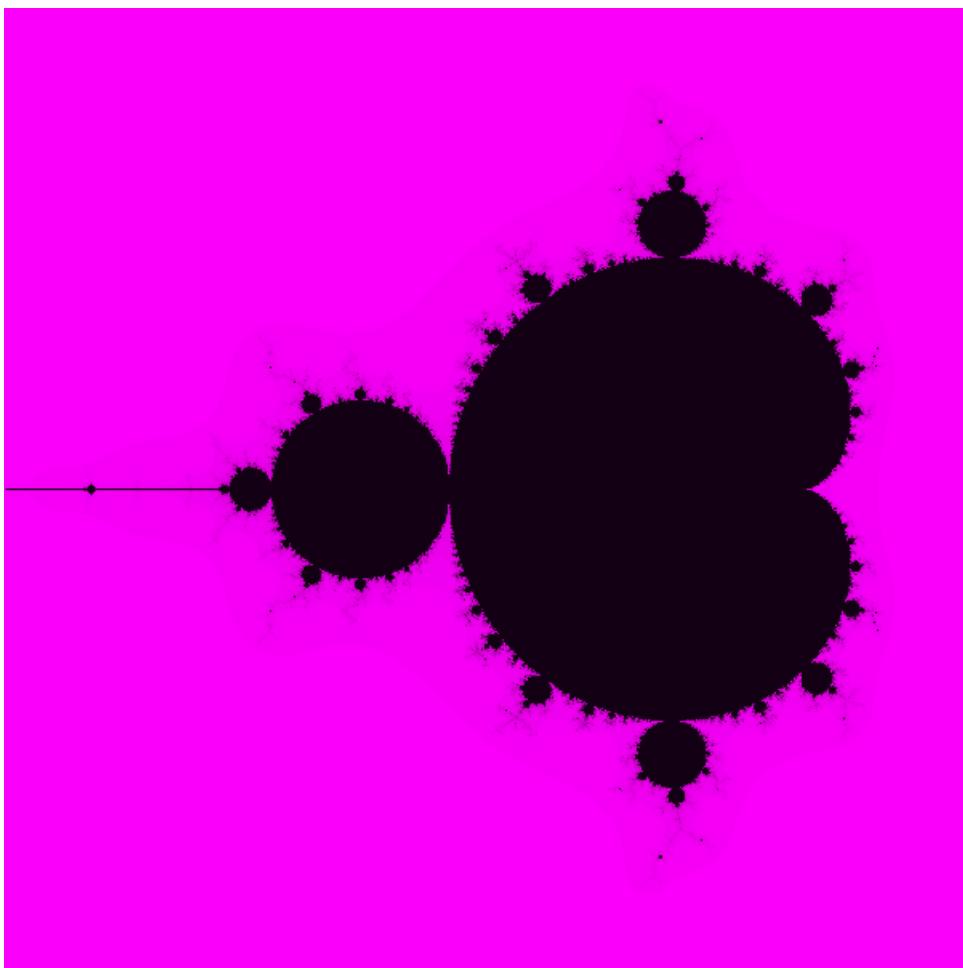
Fordítás, futtatás:



```
Változó argumentumszámú ctor — --bash — 80x24
KovachMacbookAir:Változó argumentumszámú ctor kovacsattila$ g++ main.cpp
-lpng -std=c++11
KovachMacbookAir:Változó argumentumszámú ctor kovacsattila$ ./main
Kép mérete(m*h): 360000x1411 MB
KovachMacbookAir:Változó argumentumszámú ctor kovacsattila$
```

---

És a létrejött created.png:



## Másoló-mozgató szemantika

A feladat megoldása az 5.4-ben.

## Összefoglaló (Másoló-mozgató szemantika)

Összefoglalásra az 5.3-as feladatot választottam, ami a másoló-mozgató szemantika.

C++ esetén, ha létrehozunk egy objektumot, akkor minden konstruktorkat használunk. Ez a konstruktur lehet általunk megírt is, vagy ha nem, akkor a fordító csinál nekünk egyet. Sok esetben jó az alapértelmezett konstruktur, viszont nem minden. Pl. a binfás esetben, mivel másolni akarjuk az objektumot, ezért sok esetben érdekes eredményeket kaphatunk az alap konstruktordal, amit a fordító generál nekünk. Főleg mivel az objektumunk tartalmaz pointereket is (ilyen esetekben minden kell a programozó által megírt konstruktur).

A fent említett másolás esetén a pointereket másoljuk, tehát nyílván mindenkit objektumnál ugyanarra a memóriaterületre mutatnak a mutatóink, ezért lényegében mindenkit objektum esetén ugyanott változtatunk/törlünk stb.

Ezeket a hibákat elkerülendő használunk "saját készítésű" konstruktorkat. Ezt nevezik mély(deep)másolásnak (az előbbi pedig sekély(shallow)másolásnak). Ekkor nem a pointereket, hanem a pointerek által mutatott adatokat másoljuk.

Két fajta ilyen konstruktur van: másoló és mozgató.

A másoló konstruktor lényege abban van, hogy a régi objektum adatait lemásolja az új objektum poniterei által mutatott memóriaterületekre. A két objektum tartalma ugyanaz lesz, csak lényegében a memóriaterületeik lesznek másak. Bizonyos szinten/módön ez megnyugvást hoz a programunk írása közben, mivel tudjuk, hogy bármit teszünk-veszünk, módosítunk az egyikben, az semmilyen módon nem lesz kihatással a másikra. Viszont van ennek a dolognak egy árnyoldala, mégpedig az igényelt memória megduplázódása. Mivel két külön objektumunk lesz, ezért értelemszerűen kétszer annyi helyet is fog elfolalni a memóriában.

A mozgató konstruktor, ami lényegében nem is szó szerinti másolás, már másképp működik. Ebben az esetben az történik, hogy az eredeti objektumunk pointereit kapja meg az új objektum, tehát csak az kerül átmásolásra. A régi obektum pointereit le kell nullázni. Ez nem rossz dolog, mivel memóriát takaríthatunk meg értelemszerűen. Viszont arról se feledkezzünk meg, hogy ennél a módszernél csak egy használható objektumunk marad.

Mozgató konstruktor esetén jobboldali referenciát (rvalue reference - `&&`) használunk. Arra azonban oda kell figyelnünk, hogy ilyet csak a C++11-ben, és az azt követő verziókban használhatunk. Ha terminálban fordítunk, akkor ezt a `-std=c++11/14/17` parancccsal érhetjük el.

A feladat az 5.3 esetében az volt, hogy kódcsipeteken keresztül vessük össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozzuk a mozgató értékkopírozásra!

Ehhez én a binfás progit vettettem alapul, mert ez elég jól tudja szemléltetni az ezzel a témaval kapcsolatos dolgokat. Az egész forrást nem másolom be, mert az nagyon hosszú lenne.

A lényeg az, hogy a mozgatás sokszor optimálisabb tud lenni, mint a másolás.

Pl. a binfa esetében:

```
LZWBinFa ( const LZWBinFa & regi ) {  
  
    std::cout << "LZWBinFa copy ctor" << std::endl;  
  
    gyoker = masol(regi.gyoker, regi.fa);  
}  
  
Csomopont* masol(Csomopont* cs, Csomopont* aktHely)  
{  
  
    Csomopont* uj = NULL;  
  
    if(cs != NULL){  
  
        uj = new Csomopont(cs->getBetu());  
  
        uj->ujNullasGyermek(masol(cs->nullasGyermek(),aktHely));  
  
        uj->ujEgyesGyermek(masol(cs->egyesGyermek(),aktHely));  
  
        if(cs == aktHely)  
  
            fa = aktHely;  
  
    }  
  
    return uj;  
}
```

---

Ez a másoló konstruktur lenne. Láthatjuk, hogy a masol fgv. rekurzívan fogja, és egy az egyben lemásolja az egész fát egy új objektumba. Mint fentebb is írtam, ez dupla memóriaigény.

A mozgatás már más tézszta. Itt az történik, hogy a gyökért nullpointer-re állítjuk, majd a move alap fgv.-vel átmozgatjuk a fát a régiból az újba, a régi fában pedig nem marad semmi. Mutatom a példát:

```
LZWBinFa (LZWBinFa && regi) {  
  
    std::cout << "LZWBinFa move ctor" << std::endl;  
  
    gyoker = nullptr;  
  
    *this = std::move(regi);  
  
}
```

A mozgató értékkedás pedig a swapot használja, amivel kicseréli a régi és új gyökereket:

```
LZWBinFa& operator=(LZWBinFa && regi){  
  
    std::cout << "LZWBinFa move assign" << std::endl;  
  
    std::swap(gyoker, regi.gyoker);  
  
    return *this;  
  
}
```

## 6. hét

### C++11 Custom Allocator

A feladat az volt, hogy írunk saját alloktort.

C++-ban amikor egy objektumot példányosítunk, akkor a memóriában lefoglalódik a neki szükséges terület(new operátorral).

Ez a new operator meghívja a C++ alapértelmezett allokátorát, és ez elvégzi a piszkos munkát.

Azonban mi is tudunk írni saját allokátort, ha szükségünk van rá, vagy esetleg valamilyen okból bele akarunk nyúlni a memóriakezelésbe.

Kódom a linken: <https://github.com/kovacsat2000/prog2jegyzo/blob/master/customalloc.cpp>

Nézzünk bele a kódba egy picit.

Először ott a using-os rész. Ezzel csak annyit csinálunk, hogy definiáljuk a program számára, hogy mit mivel szeretnénk helyettesíteni a továbbiakban.

Aztán jön a lényeg: az allocate. Ez a függvény egy pointert ad vissza. Ez foglalja le a memóriaterületet. Van benne egy kis követés is, kiírunk egy két dolgot, hány objektumnak foglalunk helyet hány bájton.

Ezt követi a deallokálás: deallocate. Ez egy olyan metódus, ami törli a foglalgatásainkat, felszabadítja a memóriát. Ebben is egy kis követés.

Azután a main(): létrehozunk egy int elemeket tartalmazó vektort, itt adjuk meg, hogy saját allokátort használunk.

Utáná pusbekkelünk a vektorba, és szépen sorban kiírja nekünk a progi a követéseket is.

Futásnál szépen látszik, hogy hiába pusbekkelünk pl. be egy hatodik elemet az ötödik után, akkor nem történik semmi helyfoglalás, mert ugye akkor a már az azelőtt az ötödikkel lefoglalt helyre befér a hatodik is. Viszont mikor a kilencediket pusbekkeljük, akkor újra jön a követéses-foglalásos üzenet, mivel a kilencedik már nem fér be az addig lefoglalt helyre. Tehát először egy elemnek foglal helyet, utána kettőnek, utána már a harmadik pusbekkel négynek, az ötödikkel nyolcnak, a kilencedikkkel tizenhatnak, és így tovább.

# **STL map érték szerinti rendezése**

A feladat az volt, hogy rendezzünk egy STL map-ot értékei szerint. Az adott héten talán legkönnyebb feladat volt ez.

A map-ok kules-érték párokat tartalmaznak. Felhasználásuk lényege a legtöbb esetben abban van, hogy a kulcsok rendezettsége miatt könnyen tudjuk elérni az adatokat.

Kissé hülyeségnak tünt ez az egész feladat, mivel itt az értékek szerint kellett rendezni.

Az én példámban a kulcsok számok(int), az értékek nevek(string, amik szerint majd rendezünk).

Először is létrehoztam a mapot, aminek első értéke int(a kulcs), második pedig srting(az érték - nevek), és ezeket feltöltöttem random dolgokkal.

Aztán létrehoztam a vekt vektort, ami ugye parok-at tartalmaz. Ebbe belecopyztam elejétől végéig back\_insterrel az adataimat a mapból. Azután kiiratom a vektort, ahol látszik, hogy már rendezve van kulcsértékek szerint, hiába nem rendezetten töltöttem fel. Azután lambda rendezéssel rendezem értékek szerint, és újra kiiratom.

Lentebb a screenek.

```
1 ▼ #include <iostream>
2 #include <map>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 typedef pair<int, string> parok;
9
10 ▼ int main(){
11
12 ▼     map<int, string> terk = {
13         {4, "Atti"}, {3, "Laci"}, {1, "Feri"}, {5, "Geri"}, {2, "Bandi},
14     };
15
16     vector<parok> vekt;
17
18     copy(terk.begin(), terk.end(), back_inserter(vekt));
19
20 ▼     for (auto const &parok: vekt){
21         cout << "{" << parok.first << ", " << parok.second << "}" << endl;
22     }
23
24     cout<<"-----"<<endl;
25
26 ▼     auto comp = [] (const parok& b, const parok& j) {
27         return j.second > b.second;
28     };
29
30     sort(vekt.begin(), vekt.end(), comp);
31
32 ▼     for (auto const &parok: vekt){
33         cout << "{" << parok.first << ", " << parok.second << "}" << endl;
34     }
35 }
```

```
KovachMacbookAir:stlmap kovacsattila$ g++ stl.cpp -o stl -std=c++14
KovachMacbookAir:stlmap kovacsattila$ ./stl
{1, Feri}
{2, Bandi}
{3, Laci}
{4, Atti}
{5, Geri}
{4, Atti}
{2, Bandi}
{1, Feri}
{5, Geri}
{3, Laci}
KovachMacbookAir:stlmap kovacsattila$
```

## Alternatív Tabella rendezése

A feladat az volt, hogy mutassuk be a [https://progater.blog.hu/2011/03/11/alternativ\\_tabella](https://progater.blog.hu/2011/03/11/alternativ_tabella) programban a java.lang Interface Comparable<T> szerepét.

A link alatt egy kód található, ami egy alternatív tabellát hoz létre.

Tudni kell, hogy egy sima tabella és alternatív tabella között a különbség a következő:

a sima tabella esetében a csapatok pontokat kapnak, mégpedig nyert meccsért 3 pontot, döntetlenért egy pontot, elvesztett mecsért pedig 0 pontot;

ellenben ezzel, az alternatív tabella úgy működik, hogy a csapatok aszerint kapnak pontokat, hogy ki mennyire erős csapattal játszik, és az adott meccsen milyen eredménnyel fejezi be; tehát lényegében minél erősebb csapat ellen játszik a csapat, annál több pontot pontot szerez.

A program úgy működik, hogy fordítjuk-futtatjuk a Wiki2Matrix.java-t, ez ad egy értékmátrixot, majd ezt bemésoljuk az AlternativTabella.java-ba, és ez adja vissza a kívánt alternatív tabellát.

Screen:

```
0.04393368067699879
0.05608830969430056
0.05955830154689826
0.07251615915920115
0.03295635734097101
0.05594660883459867
0.06364130248286927
0.0841168270703394
0.06360412832329729
**** Csapatokról röndezve:ellen játszik a csapat, annál több pontot pontot szerez.
A program úgy működik, hogy fordítjuk-futtatjuk a Wiki2Matrix.java-t, ez ad egy értékmátrixot, majd
| - ezt bemésoljuk az AlternativTabella.java-ba, és ez adja vissza a kívánt alternatív tabellát.
| Videoton
| 40 para
| Videoton
| 0.0841
| -
| para
| Ferencváros
| 34 Section 6.4:
| Debreceni VSC
| 0.0828
| -
```

A Csapat osztály definíciója implementálja a Comparable interfész.

A Comparable interfész arra jó, hogy össze tudjunk hasonlítani két objektumot(kisebb-e, nagyobb-e, egyenlő-e a másikkal). Egyetlen metódust tartalmaz, a compareTo-t.

Azért implementáltuk az interfész, hogy használni tudjuk annak compareTo metódusát, amivel két csapatot hasonlíthatunk össze.

A metódus objektumokat hasonlít össze(a mi esetünkben ezek a csapatok erősségei lesznek).

Esetünkben három értékkel térhet vissza: 1-el, ha a kiválasztott objektum nagyobb az objektumnál, -1-el, ha a kiválasztott objektum kisebb az objektumnál és 0-val egyébként(tehát ha egyenlők).

A kiválasztott objektum nevéhez tartozó értéket hasonlítja össze a többi névhez tartozó értékkel. A csapat osztályban létrehozott elemeket majd rendezett sorrendben adja vissza a sort függvény.

```
@SuppressWarnings({"unchecked", "rawtypes"})
default void sort(Comparator<? super E> c) {
    Object[] a = this.toArray();
    Arrays.sort(a, (Comparator) c);
    ListIterator<E> i = this.listIterator();
    for (Object e : a) {
        i.next();
        i.set((E) e);
    }
}

private static void binarySort(Object[] a, int lo, int hi, int start)
    assert lo <= start && start <= hi;
    if (start == lo)
        start++;
    for ( ; start < hi; start++) {
        Comparable pivot = (Comparable) a[start];

        // Set left (and right) to the index where a[start] (pivot)
        int left = lo;
        int right = start;
        assert left <= right;
        /*
         * Invariants:
         *   pivot >= all in [lo, left].
         *   pivot <  all in [right, start).
         */
        while (left < right) {
            int mid = (left + right) >>> 1;
            if (pivot.compareTo(a[mid]) < 0)
                right = mid;
            else
                left = mid + 1;
        }
        assert left == right;
```

## Gengszterek

Feladatunk az volt, hogy lambdával rendezzük a gengsztereket a Robotautó Világbajnokságban.

A C++11-el beköltözött a képbe egy sort függvény, ezzel kellesz dolgoznunk.

A kódcsipet:

```
std::sort (
    gangsters.begin(), gangsters.end(),
    [this, cop] ( Gangster x, Gangster y ) {
```

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

Ha ezt a sort függvényt két paraméterrel adjuk meg, akkor saját maga dönti el, hogy mi szerint rendez.

Nálunk ez a két paraméter a `gangsters.begin()` és a `gangsters.end()`, előbbi az első, míg utóbbi az utolsó elem indexét adja vissza.

Esetünkben azonban, tehát a három paraméteres verzióval van egy harmadik megadnivaló is.

Ez egy olyan függvény, ami a rendezés alapjául szolgál majd.

A példánkban látszik a szintaxis.

A lambda feltétel akkor ad majd vissza igaz értéket esetünkben, ha `x` gengszter közelebb van a rendőrhöz(`cop`), mint `y`. Tehát a sort eszerint fog rendezni.

## 7. hét

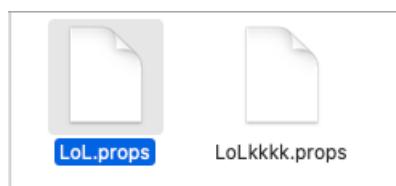
### FUTURE tevékenység editor

A feladat az volt, hogy javítsunk valamit a `ActivityEditor.java` JavaFX programon!

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

Mikor nekiugrottam ennek a feladatnak, először nem akadtam semmilyen bugra/hibára benne, de kevés nyomkodás után végül találtam.

A hiba az volt, hogy amikor átneveztünk egy tevékenységet benne, akkor a City mappában az adott helyen nem átnevezte az adott fájlt, hanem az átnevezni kívánt fájl megmaradt úgy, ahogy volt, és mellette létrehozott egy másik ÜRES fájlt, aminek a neve az volt, amire átneveztük az adott fájlt.



Ezt kiküszöbölgendő nyúlkáltam a kódban.

```
textField.setOnKeyReleased((javafx.scene.input.KeyEvent t) -> {
    if (t.getCode() == javafx.scene.input.KeyCode.ENTER) {

        String newText = textField.getText();

        java.io.File newf = new java.io.File(newText);
        java.io.File oldf = new java.io.File(oldText);
        //try {
        if (oldf.isDirectory()) {
            newf.mkdir();
        } else {
            //newf.createNewFile();
            oldf.renameTo(newf);
        }
        //} catch (java.io.IOException e) {
            //System.err.println(e.getMessage());
        //}
        commitEdit(newf);
    }
}
```

Amint látjuk, ha átneveztünk egy fájlt, akkor az futott le, hogy newf.createNewFile(), ami létrehozott egy teljesen új és üres fájlt az adott névvel.

Ezt írtam át arra, hogy átnevezze az régi(oldf) fájlt(oldf.renameTo(newf)) az általunk beírt névre.

Két érdekesség volt a dologban:

Az első az volt, hogy ha benne hagytam a try-catch-ban a dolgot, akkor nem fordult le, az a hibát adván, hogy ez a hiba sosem fog előfordulni, tehát értelmetlen a try-catch. Utánanéztem, és tényleg: a java fel tudja ismerni, hogy egy olyan kivételt akarok "elaknani", ami soha nem fog előfordulni. Ezért volt a hiba.

A másik dolog az volt, hogy a renameTo() fgv.-nek először az oldText-et akartam bedobni, mivel azt hittem, hogy stringet vár tölem. Erre hibát adott. Ennek is utánanéztem, és kiderült, hogy egy fájl változóját várja, ezért kellett az oldf változót beírni a fgv. zárójelei közé.

## OOCWC Boost ASIO hálózatkezelése

A feladat az volt, hogy mutassunk rá a scanf szerepére és használatára!

<https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

A sscanf az std:: könyvtár egy függvénye, amely formázott string inputot olvas be.

```
60     int nn = 0;
61     std::vector<Gangster> gangsters;
62
63     while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n ) == 4 )
64     {
65         nn += n;
66         gangsters.push_back ( Gangster {idd, f, t, s} );
67     }
68
69     std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( Gangster x, Gangster y ) {
70         return cop ( x, y );
71     });
72 }
```

A példánkban láthatjuk, hogy a sscanf négy paramétert vár beolvasásra. Akkor hozza csak létre az új Gangster objektumot, ha minden a négy adat beolvasása sikeres volt, és ezt betesszük a gangsters vektorba.

Amikor siker volt, az nn változóhoz mindenhol hozzáadjuk az n változó értékét. Ez majd megmutatja nekünk az összes eddig beolvasott karakter számát, ami azért kell majd nekünk, hogy tudjuk, hogy hol nem olvastunk még, és innen folytassuk az olvasást. Ezt a %n-el tesszük, ami nem egy paraméter. Ez fog számolni, az értéke átadódik n-nek, ami szépen növeli nekünk nn-t. Amint látjuk, a data-hoz van hozzáadva az nn, tehát annyit megyünk előre, amennyit eddig már beolvastunk.

## OSM térképre rajzolása

A feladat az volt, hogy debrecen térképre dobunk rá cuccokat, ennek mintájára, ahol a Tanár Úr az országba helyezte el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537>.

A feladat nálam úgy néz ki, hogy Debrecen térképén az egyetem néhány karának helyszínére kis megjelölést rakkogattan. Ezek a karok az IK, a GTK, a BTK és az MK.

A feladat megoldásához a JXMapViewViewer könyvtárat használtam.

Először IntelliJben csináltam, de utána azt is megoldottam, hogy terminálból tudjam buildelni és futtatni.

A .java és a .pom file megtalálható itt: <https://github.com/kovacsat2000/prog2files>

A feladat megoldása nem volt túl bonyolult. A program egy maven projekt.

Először létrehozom a framet, nevet, méretet adok neki, stb., beállítom a tartalmát, a mapViewer-t, ami egy JXMapViewViewer típusú objektum.

```
JFrame frame = new JFrame("Karok");
frame.getContentPane().add(mapViewer);
frame.setSize(1280, 720);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Itt négy GeoPosition típusú objektumba beleteszem az egyes karok koordinátáit, ami egy szélesség és egy hosszúság(latitude, longitude).

```
GeoPosition ik = new GeoPosition(47.54223, 21.63959);
GeoPosition gtk = new GeoPosition(47.5526, 21.60841);
GeoPosition btk = new GeoPosition(47.55392, 21.62155);
GeoPosition mk = new GeoPosition(47.53689, 21.64123);
```

Ezeket mind beleteszem egy listába. A zoomToBestFit függvénytel úgy közelítik a track lista elemeire(a karok koordinátái) a térképen, hogy minden könnyelmesen beleférjenek a képbe. A waypoints halmazba beteszem a térképen megjelenített jelölőket.

```
List<GeoPosition> track = Arrays.asList(ik, gtk, btk, mk);

mapViewer.zoomToBestFit(new HashSet<GeoPosition>(track), 0.7);

Set<Waypoint> waypoints = new HashSet<Waypoint>(Arrays.asList(
    new DefaultWaypoint(ik),
    new DefaultWaypoint(gtk),
    new DefaultWaypoint(btk),
    new DefaultWaypoint(mk)));
```

Ezeket a jelölőket egy Waypointpainter objektum segítségével fogjuk kirajzolni. Ezt a megjelenítőt beteszem egy painters listába. Jelenleg nem sok hasznát vesszük, de ha további dolgokat jelenítenénk meg, akkor ezt a painters listát bővítenénk. A painters listát egy összetett megjelenítővel rajzoljuk rá a térképre.

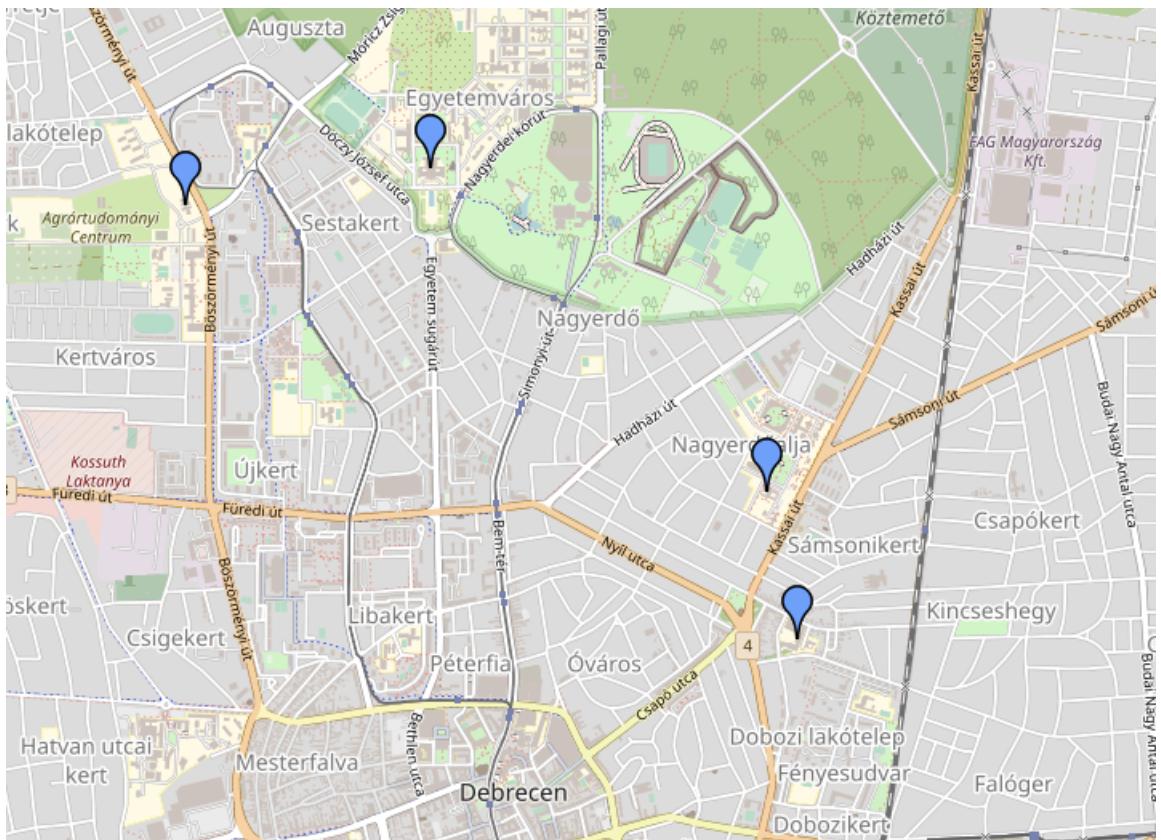
```
WaypointPainter<Waypoint> waypointPainter = new WaypointPainter<Waypoint>();  
waypointPainter.setWaypoints(waypoints);  
  
List<Painter<JXMapView>> painters = new ArrayList<Painter<JXMapView>>();  
painters.add(waypointPainter);  
  
CompoundPainter<JXMapView> painter = new CompoundPainter<JXMapView>(painters);  
mapViewer.setOverlayPainter(painter);
```

A .pom fájllal kapcsolatban még érdemes megemlíteni a dependencyt. Ezekkel a függőségekkel érem el, hogy hozzá tudjak férfi a könyvtárhoz.

```
<dependencies>  
    <dependency>  
        <groupId>org.jxmapviewer</groupId>  
        <artifactId>jxmapviewer2</artifactId>  
        <version>2.4</version>  
    </dependency>  
</dependencies>
```

És itt kellett a pluginokat is behúzni, amik arra szolgálnak, hogy tudjuk terminálból fordítani és futtatni a progit. Meg kellett adni pl., hogy milyen javat használjon, és hogy melyik legyen a mainclass.

Végül futás közben:



## BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben:

<https://github.com/nbatfai/esport-talent-search>

Az alábbi feladatban a Qt objektumokról volt szó. A Qt egy grafikai könyvtár c++-hoz.

Két fontos dolog van a képben: a signalok és a slotok.

Egy Qt objektumnak van minden kettő. A signal-t azt küldi, a slot-t pedig másik objektum signal-ját fogadja.

A signal-t és a slot-ot összekötjük, ezzel megmondjuk, hogy az adott slot melyik objektum melyik signal-ját várja.

Ezek a dolgok Qt specifikusak, tehát csak Qt-n belül érhetők el.

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

Az alábbi példában látszik, hogy a slot-okat a signal-akkal a connect fgv.-vel lehet összekapcsolni.

A programban két connect található a BrainBWin.cpp fájlban.

Láthatjuk, hogy első paraméterként a küldő szerepel(brainBThread), második paraméter a signal, a harmadik az az adott osztály, azaz this, a negyedik pedig a slot.

Első esetben a heroesChanged signal kötődik az updateHeroes slothoz. Ha a brainBThread signal-t küld a hős helyzetének megváltozásáról, akkor a slot ezt fogadja, végrehajtódik az updateHeroes fgv. és megváltoztatja a hős helyzetét.

A második pedig úgy néz ki, hogy az endAndStats kötődik az endAndStats slothoz. Ha érkezik a jel, akkor végrehajtódik az endAndStats, vége a játéknak(lejár a futási idő), és ekkor érkezik az üzenet, hogy köszöni a játékot.

```
void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &y )  
{  
  
    if ( start && !brainBThread->get_paused() ) {  
  
        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + ( this  
        * ( this->mouse_y - y );  
  
        if ( dist > 121 ) {  
            ++nofLost;  
            noFound = 0;
```

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

## 8. hét

### Port scan

A feladat az volt, hogy mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére:  
<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#id527287>

A portscan úgy működik, hogy végigmegy a parancssorban megkapott gép TCP portjain, és megpróbál kapcsolatot létrehozni.

Ha sikerül, akkor kiírja, hogy "figyeli", és be is zárjuk a socket objektumot. Ha nem, akkor kiírja, hogy "nem figyeli".

A feladatban a lényeg a try-catch volt, ami a "nem figyeli" résznél "hibakezel".

A try-ban van a végrehajtani kívánt parancssorozat, itt próbálunk kapcsolatot létrehozni. Ha sikerül, akkor lefut az egész try, és nincs semmi gond. Ha nem sikerül, akkor lép színre a catch, és 4 féle exception jöhet létre(mármint jelen esetben).

IllegalArgumentException: akkor lép fel, ha a portszám magasabb a megengedett 65535-től

IOException: akkor lép fel, ha nem sikerül socket objektumot létrehozni

UnknownHostException: akkor lép fel, ha az adott IP-hez nincs host meghatározva

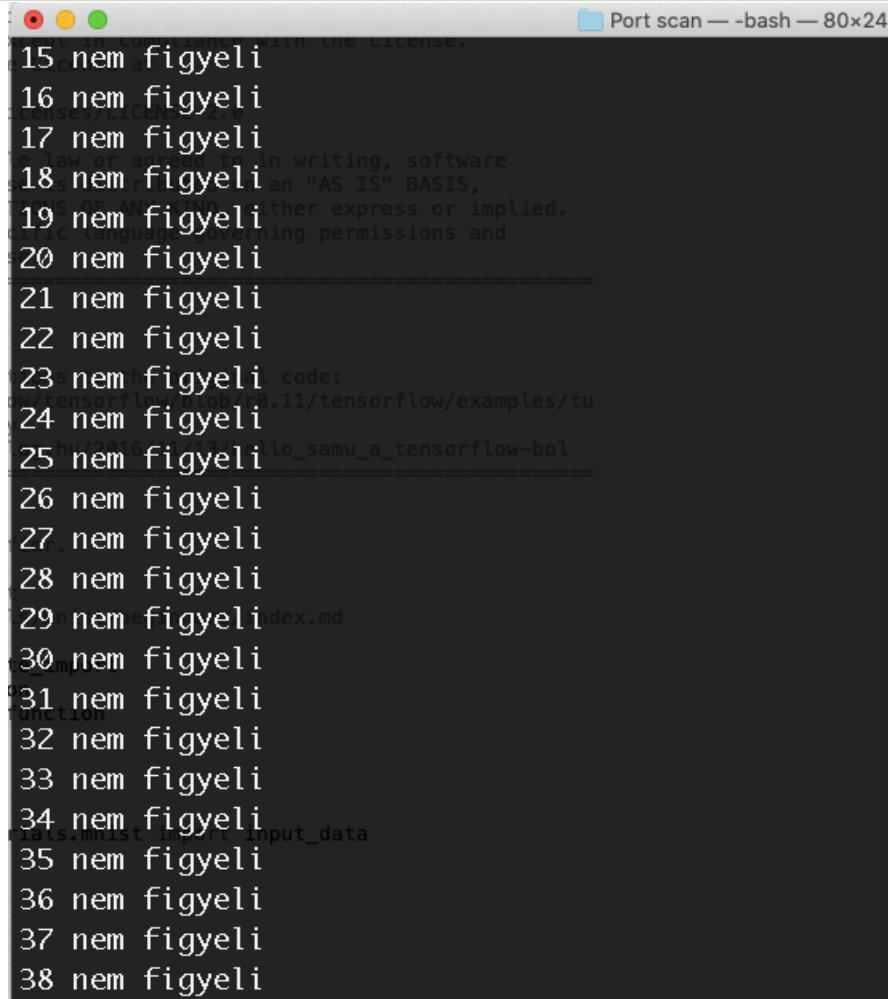
SecurityException: akkor lép fel, ha a Socket.checkConnect valamit nem akar engedni.

A try-catch egy, a programozásban sokat használt dolog. Ha nem akarjuk, hogy a programunk kipurcanjon valamilyen olyan résznél, aminek a lefutásában nem vagyunk biztosak, akkor beletesszük egy try-catch-be, és ha esetleg nem is sikerül, akkor is fut minden tovább, de megpróbáltuk út közben, amit akartunk, és még egy exception-t(kivételel) is kapunk belőle.

A kód:

```
1 ▼ public class Portscan{
2
3 ▼   public static void scanning(String a){
4       int i = 1;
5
6 ▼       while(i < 1024){
7           try {
8               java.net.Socket socket = new java.net.Socket(a, i);
9
10              System.out.println(i + " figyeli");
11
12              socket.close();
13         } catch (Exception e) {
14             System.out.println(i + " nem figyeli");
15         }
16
17         i++;
18     }
19 }
20
21 ▼   public static void main(String[] args){
22     scanning("localhost");
23   }
24
25 }
```

Futás:



The terminal window shows a port scan titled "Port scan — -bash — 80x24". The output lists numerous ports from 15 to 38, all of which are marked as "nem figyeli" (not monitored). The terminal window has a dark background with light-colored text. The title bar includes the window name and dimensions.

```
15 nem figyeli
16 nem figyeli
17 nem figyeli
18 nem figyeli
19 nem figyeli
20 nem figyeli
21 nem figyeli
22 nem figyeli
23 nem figyeli
24 nem figyeli
25 nem figyeli
26 nem figyeli
27 nem figyeli
28 nem figyeli
29 nem figyeli
30 nem figyeli
31 nem figyeli
32 nem figyeli
33 nem figyeli
34 nem figyeli
35 nem figyeli
36 nem figyeli
37 nem figyeli
38 nem figyeli
```

## Android Játék

A feladat az volt, hogy írunk egy egyszerű Androidos „játékot”!

Ehhez a feladathoz én egy, már meglévő projekt játékát választottam.

A játékom egy Android Studio használatával írt Androidos telefonokra szánt program. Gradle projekttípítő használatával készült.

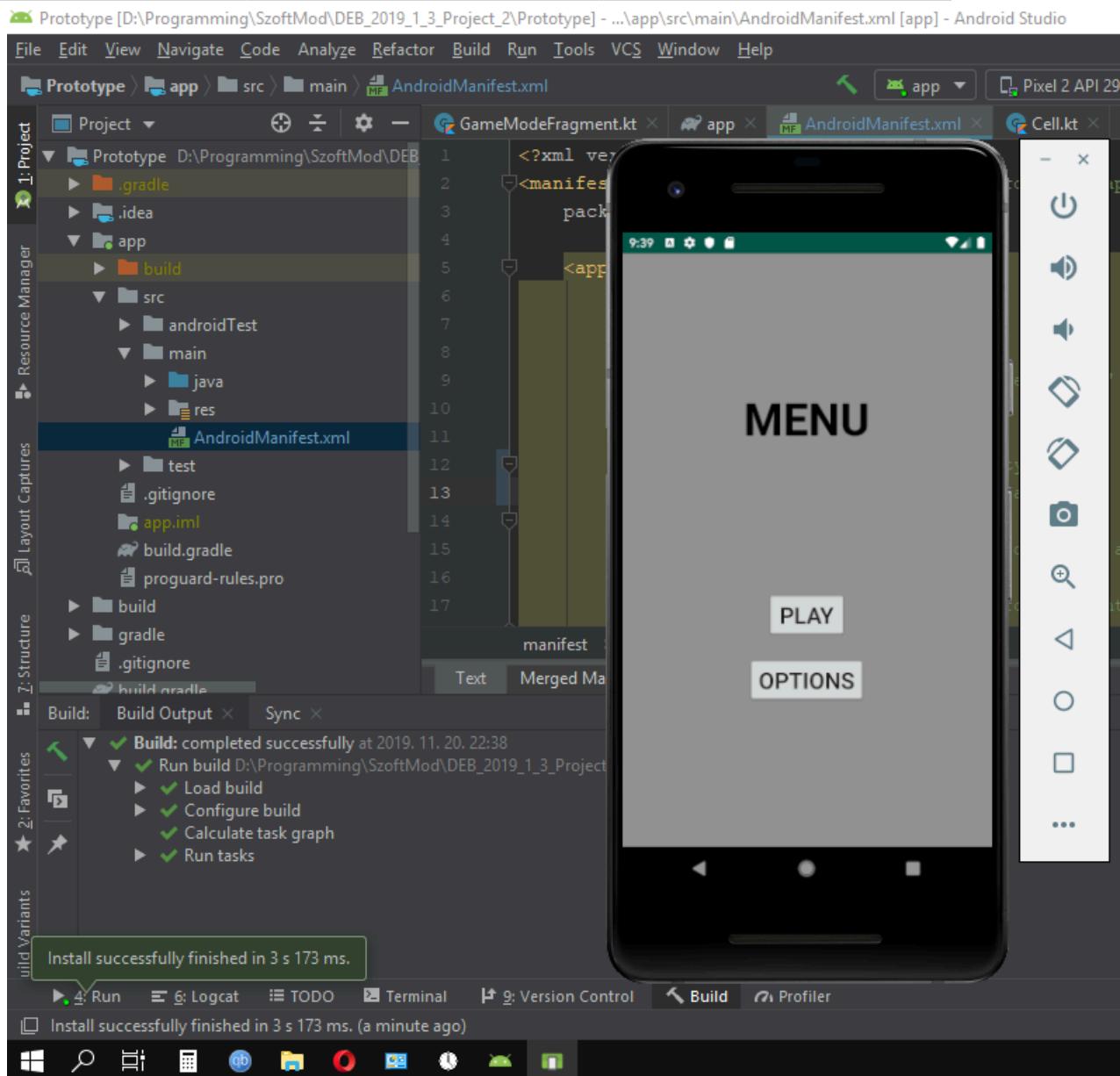
Egy monopolyhoz hasonlító kis appról van szó. Röviden összefoglalva annyi, hogy van nekünk egy karakterünk, és egy ellenfelünk. Egy tábla az alapja mindennek, amely cellákból áll.

Ezek a cellák tartalmaznak bizonyos ingatlanokat, tulajdonokat, melyeknek van egy áruk és egy körönkénti profituk. Mindkét játékos egy alaptökével indul, ezt lehet növelni/csökkenteni. Ha az adott játékos egy mezőre lép, akkor eldöntheti, hogy megveszi-e, vagy sem. Ha megveszi, akkor az ő tulajdona lesz az ingatlan, amely körönként bizonyos profitot termel.

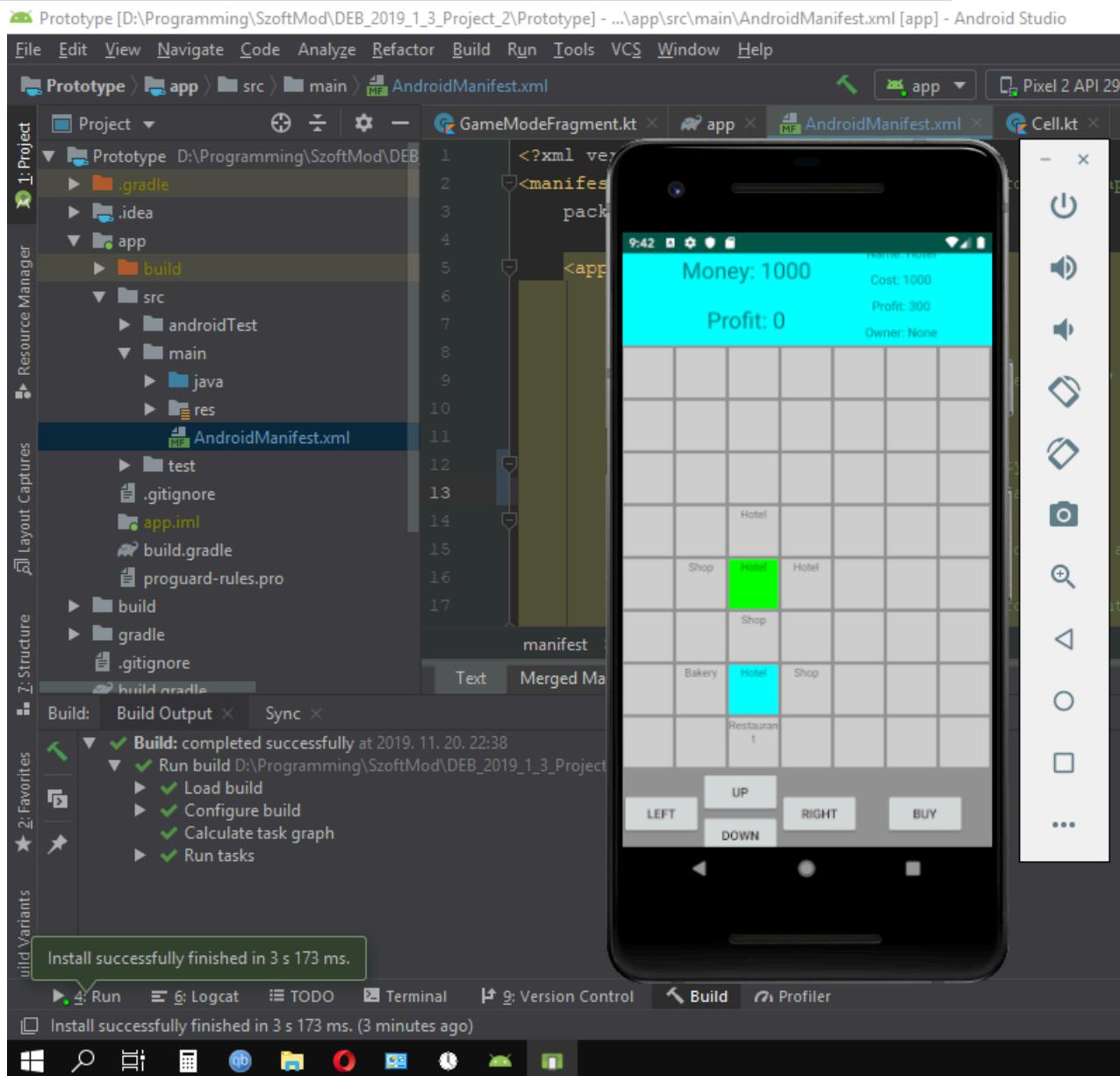
Sorasan lépünk, tehát egyszer egyik játékos, egyszer másik. Egy lépés VAGY vásárlást jelent, VAGY mozgást a cellák között.

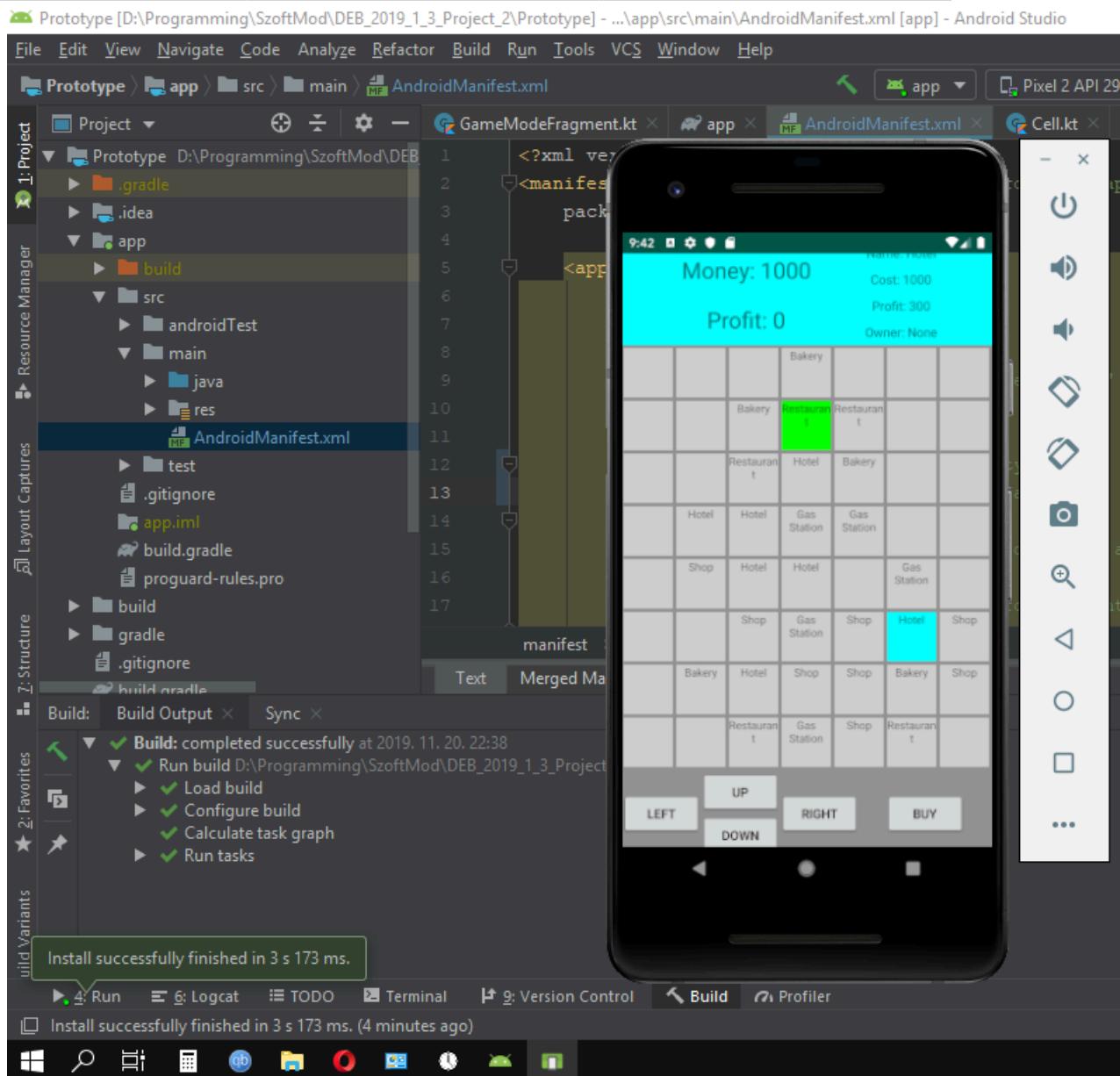
Játékmódtól függően a játék végezhet elfogyott pénzzel, lejárt idővel, lejárt lépésszámmal stb.

Így néz ki:



Magas szintű programozási  
nyelvek 2. Jegyzőkönyv.  
Kovács Attila Patrik.





A játék sok részből épül fel, mélyen nem fogok belemenni a részletekbe, de néhány dolog/érdekességet ismertetek vele kapcsolatban.

4 fregmense van:

Menu Fragment(ez a start), itt választhatunk a PLAY és az OPTIONS között

Play Fragment, itt a Game Modeok között vélogatunk

Choose Fragment, itt állíthattuk, hogy hány játékos vesz részt a játékban

és végül a Game Fragment, ez már maga a játék.

Ez mind Frontendes dolog, a Backend lényegi része mind a GameModeFragment.xml-ben található, maga a játékalgoritmika, ezekről nem írok, mert hosszú lenne, és talán annyira nem is érdekes.

Ami még a fejlesztés közben érdekes volt számomra, az a navigation(.xml).

A játék futás közben itt ugrál egyik fregmensből a másikba, itt hívódnak az action-ök. Sokat tanultam és tapasztaltam új dolgokat ennek fejlesztése közben. Sok hibába is ütköztem, de minden hasznosnak találom, sok mindenre megtanítottak.

Másik érdekesség, hogy ilyen Gradle projektek esetében rengeteg olyan fájl(többnyire .java) van, amit látunk ám fejlesztés közben, viszont ez minden "átgenerálódik" a futás közben. Ezek is sok fejtörést okoztak, mivel ezekben lett volna a legkönnyebb nyúlkálni, viszont ez nem célravezető, mivel semmi értelme az "újragenerálódás" miatt.

Még egy utolsó érdekesség a projekttel kapcsolatban: a fregmensek közötti paraméterek átadása.

Ezek is okoztak kisebb-nagyobb fejtöréseket, mivel nehezebb használni őket, mint ahogy elsőre gondolná az ember. Kifejezen nagyon nem szeretném, de annyit róluk, hogy ilyen esetben első lépésként létrehozunk egy argument-et a "destination" fregmensre a navigation.xml-ben, innen majd tudja a navi, hogy itt valaminek történni kell. Aztán a kezdőfregmens activityében "adnunk" kell a paramétert, és a cél fregmensben pedig fogadnunk kell. Elsőre talán egyszerűnek hangzik, de korántsem volt az.

## JUnit teszt

A feladat az volt, hogy a [https://progpater.blog.hu/2011/03/05/labormeres\\_oththon\\_avagy\\_hogyan\\_dolgozok\\_fel\\_egy\\_pedat](https://progpater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_egy_pedat) poszt kézzel számított mélységet és szórását dolgozzuk be egy JUnit tesztbe.

Először is mi a JUnit test. A JUnit test egy olyan keretrendszer, amit egységesztelésre használnak Java nyelv mellé.

A példánk így néz ki:

```
1 import static org.junit.Assert.*;
2 import org.junit.Test;
3
4 public class BinfaTest {
5     LZWBinFa binfa = new LZWBinFa();
6
7     @org.junit.Test
8
9     public void tesBitFeldolg() {
10         for (char c : "01111001001000111".toCharArray())
11         {
12             binfa.egyBitFeldolg(c);
13         }
14         org.junit.Assert.assertEquals(4, binfa.getMelyseg(), 0.0);
15         org.junit.Assert.assertEquals(2.75, binfa.getAtlag(), 0.001);
16         org.junit.Assert.assertEquals(0.957427, binfa.getSzoras(), 0.0001);
17     }
18 }
```

Először is nézzük a 7. sort. Ott a @-al kezdődő sor jelöli azt a metóduskezdetet, amit a tesztfuttatónak futtatnia kell. Egy teszeset kezdődik kukac annotációval, aminek a törzsében lévő metódus kerül meghívásra. Itt dől el, hogy a kapott eredmény azonos-e az elvárt eredménnyel.

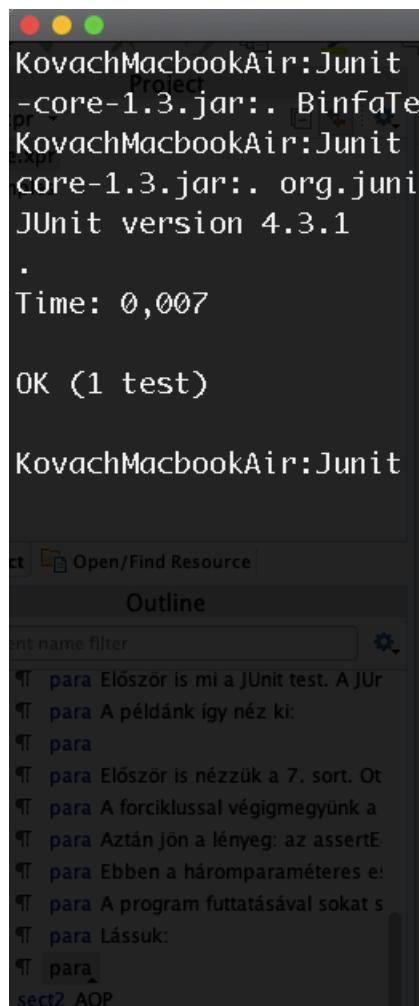
A forciklussal végigmegyünk a karaktersorron, ami a binfa objektumba karakterenként dolgozza bele(egyBitFeldolg()) az adatokat, tehát itt töltödik fel a binfa.

Aztán jön a lényeg: az assertEquals fgv.-vel vizsgáljuk meg, hogy az adott fgv.-k jól működnek-e.

Ebben a háromparaméteres esetben az első paraméter jelenti azt, hogy mit várunk vissza, a második paraméter az a tesztelni kívánt metódus, a harmadik pedig a hibahatárt határozza meg.

A program futtatásával sokat szívtam, nem akarta találni a JUnit-os importokat(mivel meg se voltak igazából), de végül egy letöltéssel és egy fordítás közbeni importtal sikerült ezt is megoldani.

Lássuk:



```
Junit teszt — bash — 88x24
KovachMacbookAir:JUnit teszt kovacsattila$ javac -cp /Library/Junit4/junit-core-1.3.jar:. Binfatest.java
KovachMacbookAir:JUnit teszt kovacsattila$ java -cp /Library/Junit4/junit-core-1.3.jar:. org.junit.runner.JUnitCore Binfatest
JUnit version 4.3.1
.
Time: 0,007
OK (1 test)

KovachMacbookAir:JUnit teszt kovacsattila$
```

Eloszor is nezzük a 7. sort. Ott a @-ai kezdődő sor jelöli azt a metóduskezdetet, amelyet futtatnia kell. Egy teszeset kezdődik kukac annotációval, aminek a törzsébe van elhelyezve. Itt dől el, hogy a kapott eredmény azonos-e az elvárt eredménnyel.

A forciklussal végigmegyünk a karaktersrövidítésekkel, ami a binfa objektumban minden karaktert meghatározza. Aztán jön a lényeg: az assertEquals fgv.-vel vizsgáljuk meg, hogy mit vártunk és mit kapottunk. Ebben a háromparaméteres esetben az első paraméter jelenti azt, hogy mit vártunk, a második pedig a hármat, a harmadik pedig a hibahatárt határozza meg.

A program futtatásával sokat szívtam, nem akarta találni a JUnit-os importokat, de végül egy letöltéssel és egy fordítás közbeni importtal sikerült ezt is megoldani.

Lássuk:

para

para

## Section 8.4: AOP

para

## AOP

A feladat az volt, hogy szöjünk bele egy átszövő vonatkozást az első védési program(binfa) Java átitratába!

Ennél a feladatnál az aspektusorientáltságról van szó.

Van egy alapnyelvünk, jelen esetben ugye Java(komponensnyelv). Ehhez jön az aspektusnyelv, ami megegyezhet, vagy akár el is térhet a komponensnyelvtől. Tehát a komponensnyelven megírt programunkra tudunk aspektusokat alkalmazni, ami megváltoztatja a programunk működését. Ez maga a szövés.

Én a következőt tettem:

```
1 public aspect Counting {  
2  
3     private int cnt = 0;  
4  
5     before(char c) : call(public void egyBitFeldolg(char)) && args(c) {  
6  
7         cnt++;  
8  
9         System.out.println(cnt);  
10    }  
11}  
12  
13}
```

A kis átszövés azt csinálja, hogy figyeli, hogy mikor hívódik meg az egyBitFeldolg() fgv., és akkor növeli a változót, és ki írja azt. Tehát sorban ír ki számokat, és közük az utolsó megmutatja, hogy hányszor hívódott meg az adott fgv.-nk.

Futás:

The screenshot shows a terminal window titled "AOP — -bash — 88x24". The command entered is "java jegyzokonyvF.xml". The output of the program is displayed, showing the count of character 'c' being printed each time it is called, followed by the final count.

```
258  
259  
260  
261  
262  
263  
264     1 public aspect Counting {  
265     2  
266     3     private int cnt = 0;  
267     4  
268     5     before(char c) : call(public void egyBitFeldolg(char)) && args(c) {  
269     6         cnt++;  
270     7  
271     8         System.out.println(cnt);  
272     9     }  
273    10  
274    11     }  
275  
276  
277  
278  
279  
280  
En a következőt tettek:  
▶  
▶  
▶  
▶  
1  public aspect Counting {  
2  
3     private int cnt = 0;  
4  
5     before(char c) : call(public void egyBitFeldolg(char)) && args(c) {  
6         cnt++;  
7  
8         System.out.println(cnt);  
9     }  
10    }  
11  
12  
13 }  
144  
A kis átszövés azt csinálja, hogy figyeli, hogy mikor hívódik meg az egyBitFeldolg() fgv., és akkor változót, és ki is írja azt. Tehát sorban ír ki számokat, és közük az utolsó megmutatja, hogy hány hívódott meg az adott fgv.-nk.  
Futás:  
para  
KovachMacbookAir:AOP kovacsattila$
```

A legtöbbet talán magával a fordítással és a futtatással vesződtem(plusz a telepítés és a pathok beállítása sem volt egyszerű), de végül sikerült.

## 9. hétköznap

### MNIST

A feladat az az alap feladat megoldása, + saját kézzel rajzolt képet is ismerjen fel.

Az MNIST egy gépi tanulást modellező alkalmazás. Betanul, és képeken lévő számokat ismer fel.

Ez egy tensorflow nevezetű könyvtár része, amely a gépi tanulást segíti elő.

Az mnist feladata, hogy felismerjen egy beolvasott képen lévő számot, amit a readimg() fgv.-vel olvas be.

Ahhoz, hogy fel tudjon ismerni számokat, előbb be kell tanítanunk a programot. Ez úgy működik, hogy a megadott modellen betanul a progi, majd egy pontosság tesztelés során becsül egy elméleti pontosságot a "tudásáról".

Miután betanul, először a W\_0 súlyokat ábrázolja, ez mondja meg a 0 valószínűségét. Az ebből jövő ábra mutatja meg, hogy a progi milyennek gondolja a 0-t.

Az MNISTnél is - mint már korábban - egy neurális hálót használunk. Ennek vannak rétegei, amelyek az inputból információkat szereznek.

Vátozókat tekintve: van egy x, ami egy tenzor, amibe az értékeket küldjük, ő tudja azt is, hogy 28x28-as képet kell felismernie; a w egy súly, b - bias, valószínűség.

A célunk a tanítás során az, hogy minimalizáljuk az eredeti és a becsült érték közötti különbséget. A programban felüli a fejét egy crossentropy nevezetű fgv., ami pont ezt a különbséget mutatja meg nekünk.

Ezeket követi a tanulási fázis a GradientDescent és a SoftMax optimalizációs függvény segítségével.

Most pedig jöhét a felismerés:

Én két képet ismertetek fel vele: egy mnistes képet, ami egy 4-es, egy saját kézzel írt számot, ami egy 8-as lett, és egy szintén saját 3-ast.

A gépemen a betanulás viszonylag gyorsan megy, kb olyan 6-8 mp alatt kész is, eztán a megadott számokat ábrázoló képeket ismertetem fel a progival.

A progi 28x28-as képet vár, ezért a saját számaimat át kellett méretezni.

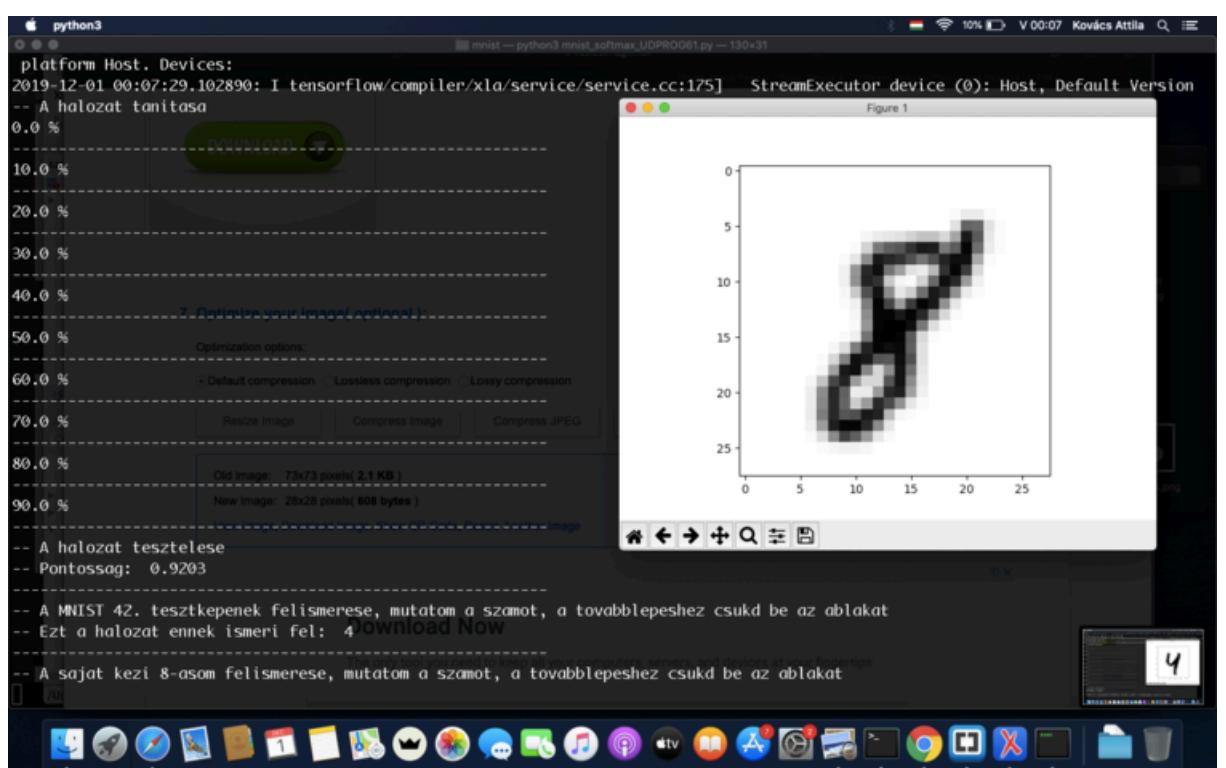
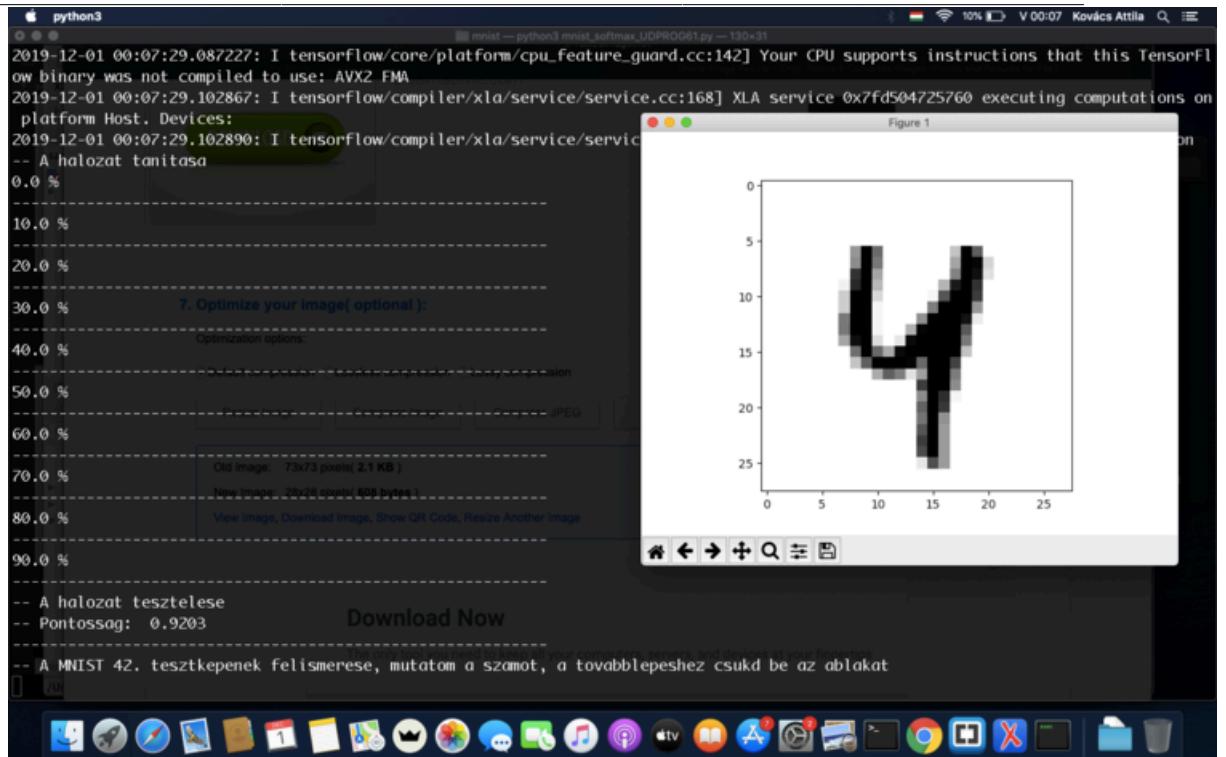
A program elindításával rengeteg sok időt szenvédtem, mivel mostanra a gépem kb. egy nagy katyvasz, ráférne már egy op.rendszer újratelepítés. Az egéssel az a baj, hogy mindenféle könyvtárak ömlesztve vannak feltelepítve rá, általában nem is egy, hanem két-három verzió mindenből, és az ilyen dolgok miatt állandóan valamilyen problémába ütközök, amik általában olyanok, hogy nem talál valamilyen könyvtárat, vagy rossz elérési utvonallal miatt sér.

A tensorflowt telepítettem, hogy tudjam futtatni a progit. Ezen kívül egy virtuális környezetben futtatom a programot, mert sok helyen ajánlják, hogy tensorflow esetén érdemes.

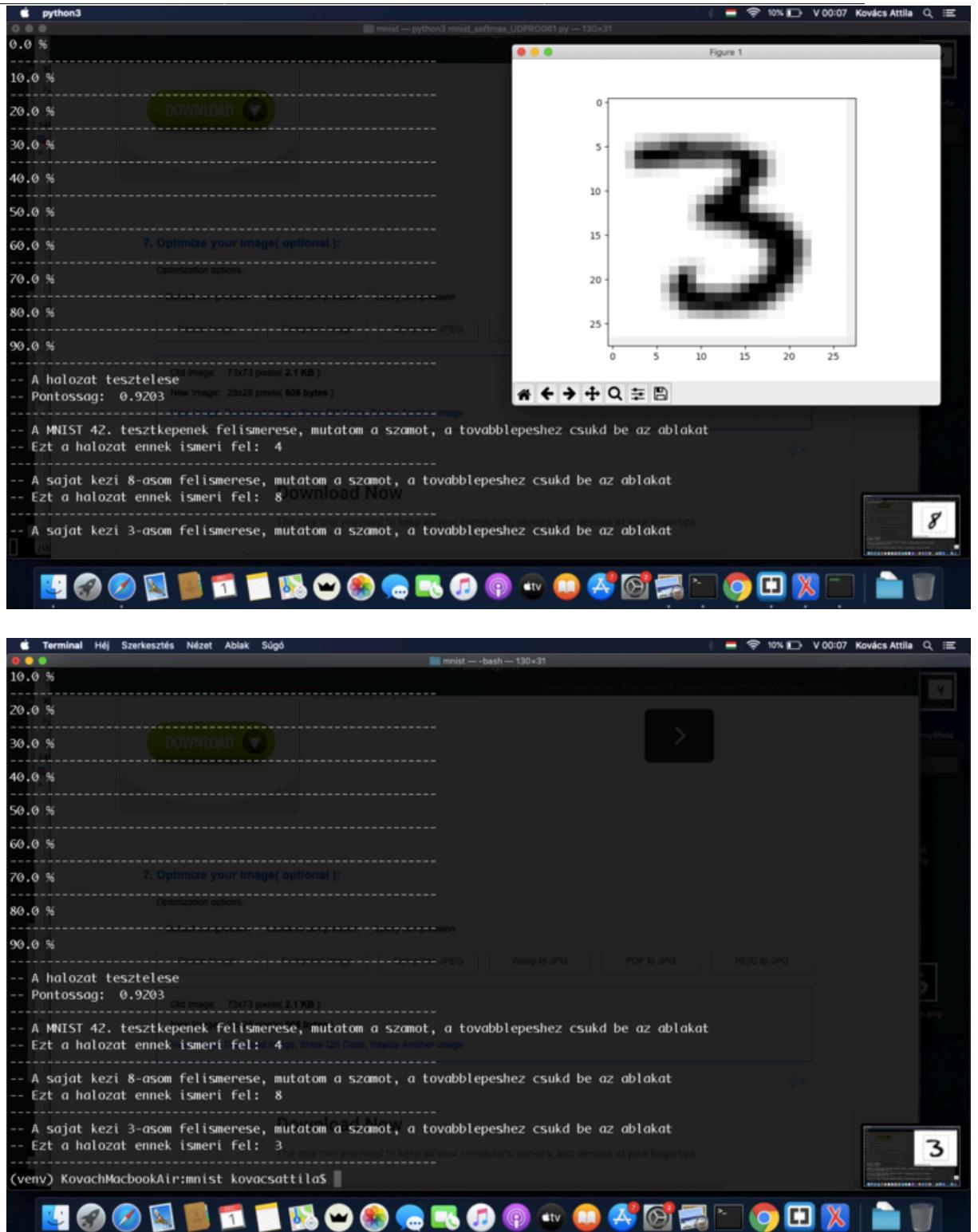
De kb. ilyen 6-8 óra kemény szenvédés, hiba-keresés és hiba-javítgatás után feléled a program. Természetesen nagyon örültem neki.

Lássuk:

Magas szintű programozási  
nyelvek 2. Jegyzőkönyv.  
Kovács Attila Patrik.



Magas szintű programozási  
nyelvek 2. Jegyzőkönyv.  
Kovács Attila Patrik.



## Deep MNIST

A feladat az az, mint a fenti MNIST, de a mély változattal.

A feladat nagyon hasonlított az előző feladathoz. A különbség az volt, hogy mivel ez mély változat, ezért több rétegen mennek át az adatok, mélyebben tanul be a program.

A kód részletek:

```
def readimg():
    file = tf.read_file("sajat8as.png")
    img = tf.image.decode_png(file, 1)
    return img

def readimg1():
    file = tf.read_file("sajat3as.png")
    img = tf.image.decode_png(file, 1)
    return img
```

Itt ugyanúgy beolvassuk a képet a mappánkból(saját kép). És itt dekódoljuk a képet úgy, hogy a kimeneti color channelje gray scale legyen.

```
# Training steps
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

saver = tf.train.Saver()

"""max_steps = 1000
for step in range(max_steps):
    batch_xs, batch_ys = mnist.train.next_batch(50)
    if (step % 10) == 0:
        print("Iter: ", step, " --- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.
            train.images, y_: mnist.train.labels, keep_prob: 1.0}))
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys, keep_prob: 0.5})
print("Iter: " max_steps, " --- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.
    train.images, y_: mnist.train.labels, keep_prob: 1.0}))
saver.save(sess, "model.ckpt")"""

saver.restore(sess, "model.ckpt")
```

Ez pedig a tanítási rész. Itt elég sokáig tartott a tanítás, mivel 20000 iteráció megy keresztül, és maga a modell is eléggé bonyolult.

Azért van kimentve a tanítási állapot, hogy ne kelljen minden futtatásnál kivárni a legalább 20-25 percet, amíg tanul. (A példámban azért megy 1000-ig az iteráció, mert mikor először futtattam 20000-ig, akkor még nem mentettem ki a tanultakat, és másodszor nem volt türelmem megint kivárni a 20000-et)

A kimentés egyébként a saver.save()-el történik, a kimentettek használata pedig a saver.restore()-al.

Végül a tesztelés:

```
img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image], keep_prob: 1.0})
print("-- Az MINST 42. képenek felismerése")
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])

img = readimg()
image = img.eval()
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image], keep_prob: 1.0})
print("-- A saját kezi 8-asom felismerése, mutatom a szamot, atovabblepeszhez csukd be")
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])

img = readimg1()
image = img.eval()
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image], keep_prob: 1.0})
print("-- A saját kezi 3-asom felismerese, mutatom a szamot, atovabblepeszhez csukd be")
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
```

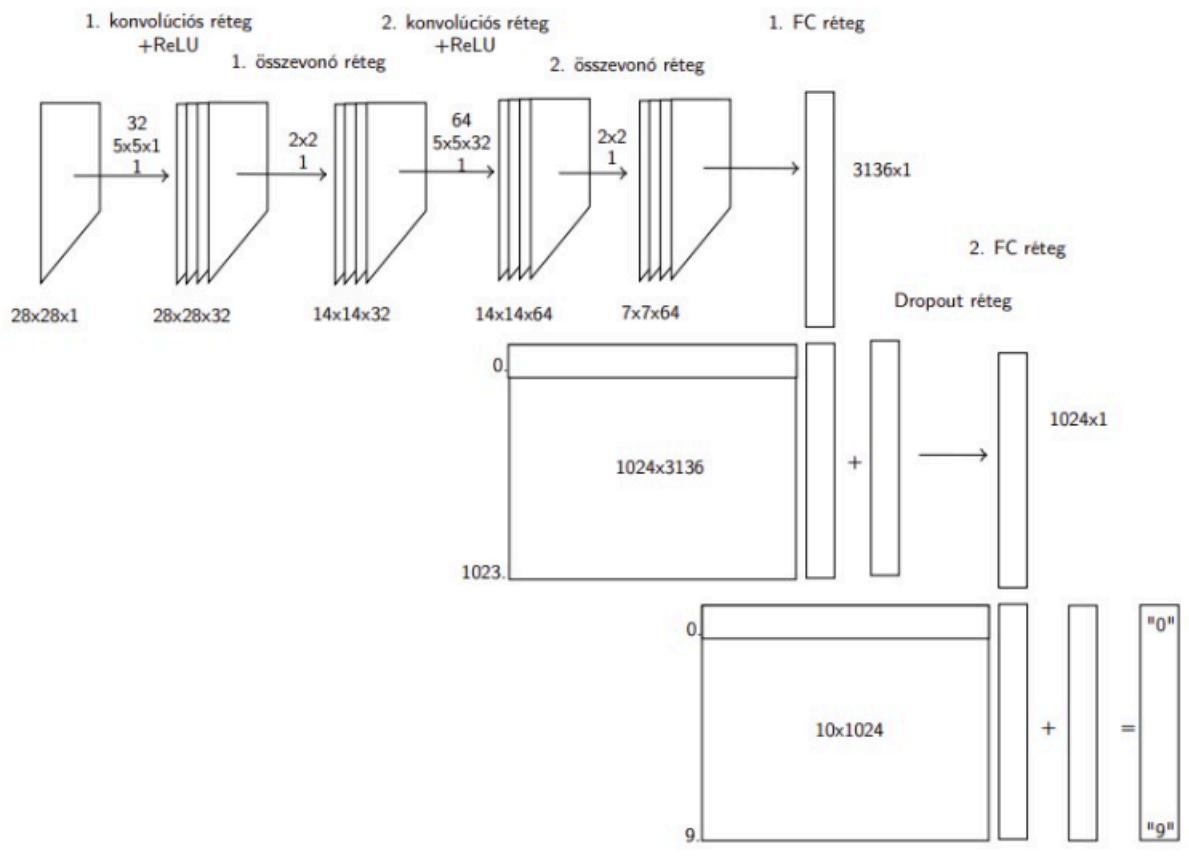
3 képet tesztelek, 1 alapot és 2 sajátot. Ami engem is megtévesztett, az az argmax-ban az y változó.  
Az előzőben ez y\_ volt, de a deepmnistben már sima y néven szerepel, át kellett írni.

A kiértékelt y(\_conv)-ból kiválasztjuk a legnagyobb értékkel rendelkező indexet, ami a legfontosabb illeszkedés.

Futás:

```
Iter: 850 --- Pontossag: 0.9577
Iter: 860 --- Pontossag: 0.959
Iter: 870 --- Pontossag: 0.9601
Iter: 880 --- Pontossag: 0.9581
Iter: 890 --- Pontossag: 0.9584
Iter: 900 --- Pontossag: 0.9584
Iter: 910 --- Pontossag: 0.9598
Iter: 920 --- Pontossag: 0.9575
Iter: 930 --- Pontossag: 0.9569
Iter: 940 --- Pontossag: 0.9587
Iter: 950 --- Pontossag: 0.959
Iter: 960 --- Pontossag: 0.9592
Iter: 970 --- Pontossag: 0.9589
Iter: 980 --- Pontossag: 0.9587
Iter: 990 --- Pontossag: 0.9616
1000 0.9573
-- Az MINST 42. képenek felismerése
-- Ezt a halozat ennek ismeri fel: 4
-- A saját kezi 8-asom felismerése, mutatom a számot, atovabblepeshet
-- Ezt a halozat ennek ismeri fel: 8
-- A saját kezi 3-asom felismerése, mutatom a számot, atovabblepeshet
-- Ezt a halozat ennek ismeri fel: 3
(base) KovachMacbookAir:mnist kovacsattila$ ls
```

Tudatja velünk minden 10. iteráció után a pillanatnyi pontosságot. Ez itt az 1000. iterációig felmegy olyan 0.95-0.97 közé, de amikor először futtattam 20000-ig, akkor felment 1-re.



A modell bonyolult, és ezt a fenti ábra is szemlélteti.

Az első réteg a bemenetként átadott kép.

Az első konvolciós réteg a köv. képpen jön létre:

with tf.name\_scope('conv1'):

```
' W_conv1 = weight_variable([5, 5, 1, 32])
' b_conv1 = bias_variable([32])
' h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

A conv2d a bemenő paramétereknek számolja ki a kétdimenziós konvolcióját.

with tf.name\_scope('conv2'):

```
' W_conv2 = weight_variable([5, 5, 32, 64])
' b_conv2 = bias_variable([64])
' h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
```

with tf.name\_scope('pool2'):

```
' h_pool2 = max_pool_2x2(h_conv2)
```

```
with tf.name_scope('fc1'):  
  
' W_fc1 = weight_variable([7 * 7 * 64, 1024])  
  
' b_fc1 = bias_variable([1024])  
  
' h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])  
  
' h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

---

```
with tf.name_scope('fc2'):  
  
W_fc2 = weight_variable([1024, 10])  
  
' b_fc2 = bias_variable([10])  
  
' y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2  
  
return y_conv, keep_prob
```

A max\_pool maximum poolozást csinál. Ezt követi egy újabb konvolciós réteg. Ezután létrejön a fullyconnected réteg. Majd bevezetünk baisokat. És 10 értékre képezzük le a súlyokat és biasokat.

## CIFAR-10

A feladat az az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel.

Az ezelőtti MNIST-es példákban számokat ismertettünk fel a programmal. Most a Cifar-10 esetén viszont már mászt a helyzet. Ez a program színes 32x32-es képeket sorol be kategóriákba(pl. repülő, kutya, macska, auto, stb.).

A progi alap forrása megtalálható a tensorflow mappájában, viszont a mi feladatunk esetén jócskán bele kellett nyúlkálni a forrásokba.

Kezdjük is ott, hogy saját képet kell vele felismertetni. A program a képet bináris formában olvassa, ezért a saját képünket át kell alakítani:

```
from PIL import Image  
import numpy as np  
import sys  
  
im = Image.open(sys.argv[1])  
im = (np.array(im))  
  
r = im[:, :, 0].flatten()  
g = im[:, :, 1].flatten()  
b = im[:, :, 2].flatten()  
  
label = [0]  
  
out = np.array(list(label) + list(r) + list(g) + list(b), np.uint8)  
out.tofile("./cifar10_data/cifar-10-batches-bin/" + sys.argv[2])
```

Először is RGB-re bontjuk a bemenetet, amiket tömbben tárolunk. Aztán ezeket felfűzzük listába, és bájttá alakítjuk. Kimentjük egy .bin fájlba.

A cifar10\_input.py-ba a következőt kellett módosítani:

```
filenames = [os.path.join(data_dir, 'input.bin')]
```

Ez az input.bin általunk létrehozott fájlból olvassa bemenetnek a bájtokat.

Emellett pedig át kell állítani a batch size-t 1-re, mivel 1 képet ismertetünk fel vele:

```
tf.app.flags.DEFINE_integer('batch_size', 1, """Number of images to process in a batch."""")
```

Az eredeti forrás pontosságot is mutat, de ezt kivettük, és csak a kategóriákba való besorolást kérjük cifartól:

```
#while step < num_iter and not coord.should_stop():
predictions = sess.run([top_k_op])

print(sess.run(logits[0]))
classification = sess.run(tf.argmax(logits[0], 0))
ifar10classes = ["airplane", "automobile", "bird", "cat", "deer", '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))
```

Még egy fgv. hívásnál és írásnál be kellett biggyesztünk a logits tömböt is argumentumként.

## Android telefonra a TF objektum detektálója

A feladat az volt, hogy telepítsük fel és próbáljuk ki!

A program tárgyat ismert fel a telefon kamerájának segítségével.

Ezzel a feladattal viccesen az volt a baj, hogy nem tudtam min kipróbálni, mivel nincs andoidos telefonom, és a családban sincs senkinek. Viszont tavaly kipróbáltam a szobatársam telefonján, mikor jegyzőkönyvet csináltam, és csodával határos módon a screenshotokat megtaláltam a driveomban. Akkor úgy csináltam, hogy importoltam a githubról klónolt repót andr. studioba, és létrehoztam egy .apk-t belőle. Ezt feltelepítettem a telefonra és már szuperált is. Íme:

loudspeaker: 0.50028527



fountain pen: 0.1898813  
hammer: 0.17330307  
plane: 0.103508964



mouse: 0.16615319

lens cap: 0.13163683



Magas szintű programozási  
nyelvek 2. Jegyzőkönyv.  
Kovács Attila Patrik.

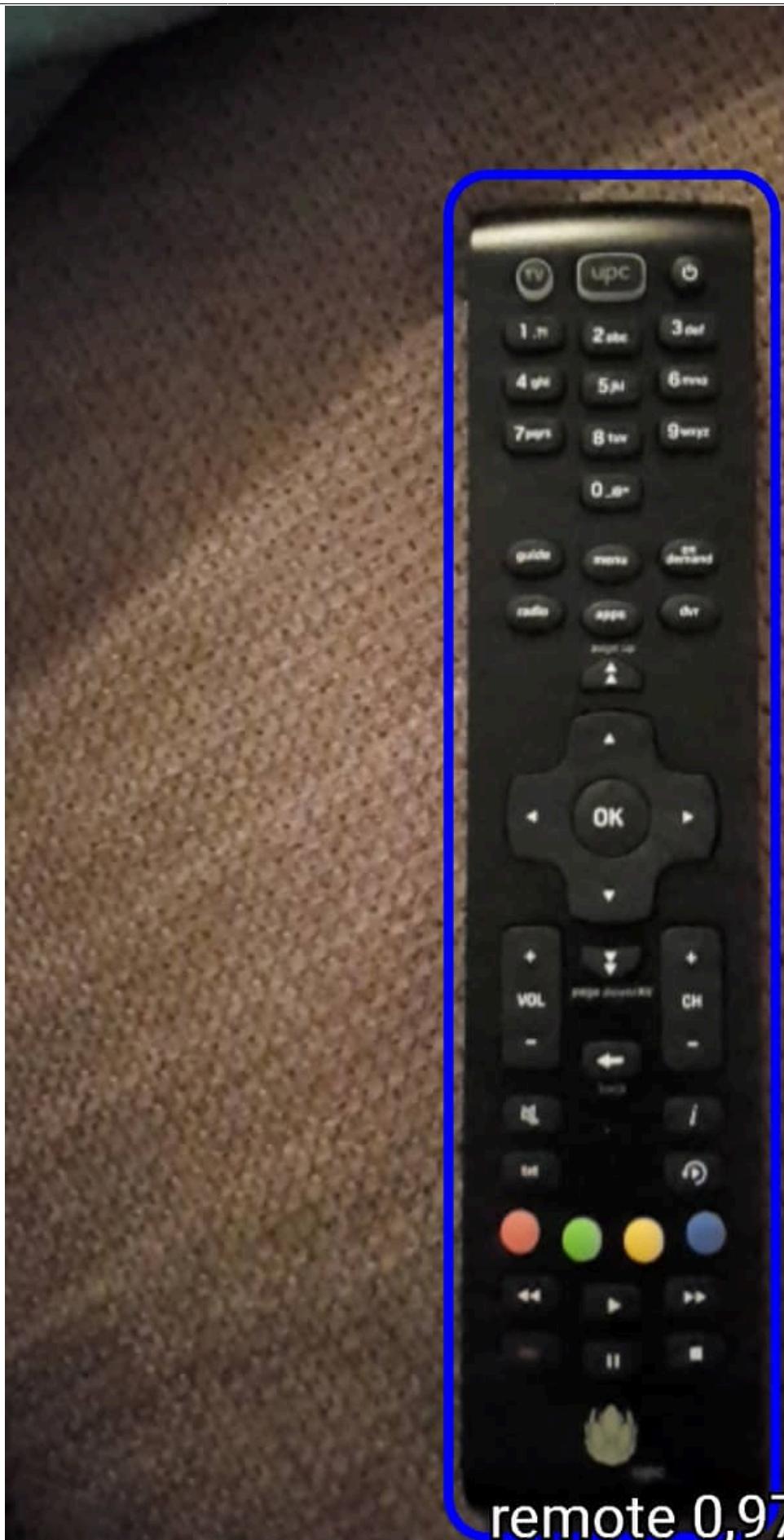
---

Nagyon érdekesnek találtam ezt a kis appot, és plusz pozitívum, hogy szívni se kellett vele sokat.

(3 nappal később):

Egy barátom telefonjára ismét frissen feltelepítettem az .apk-t. Kipróbáltuk:

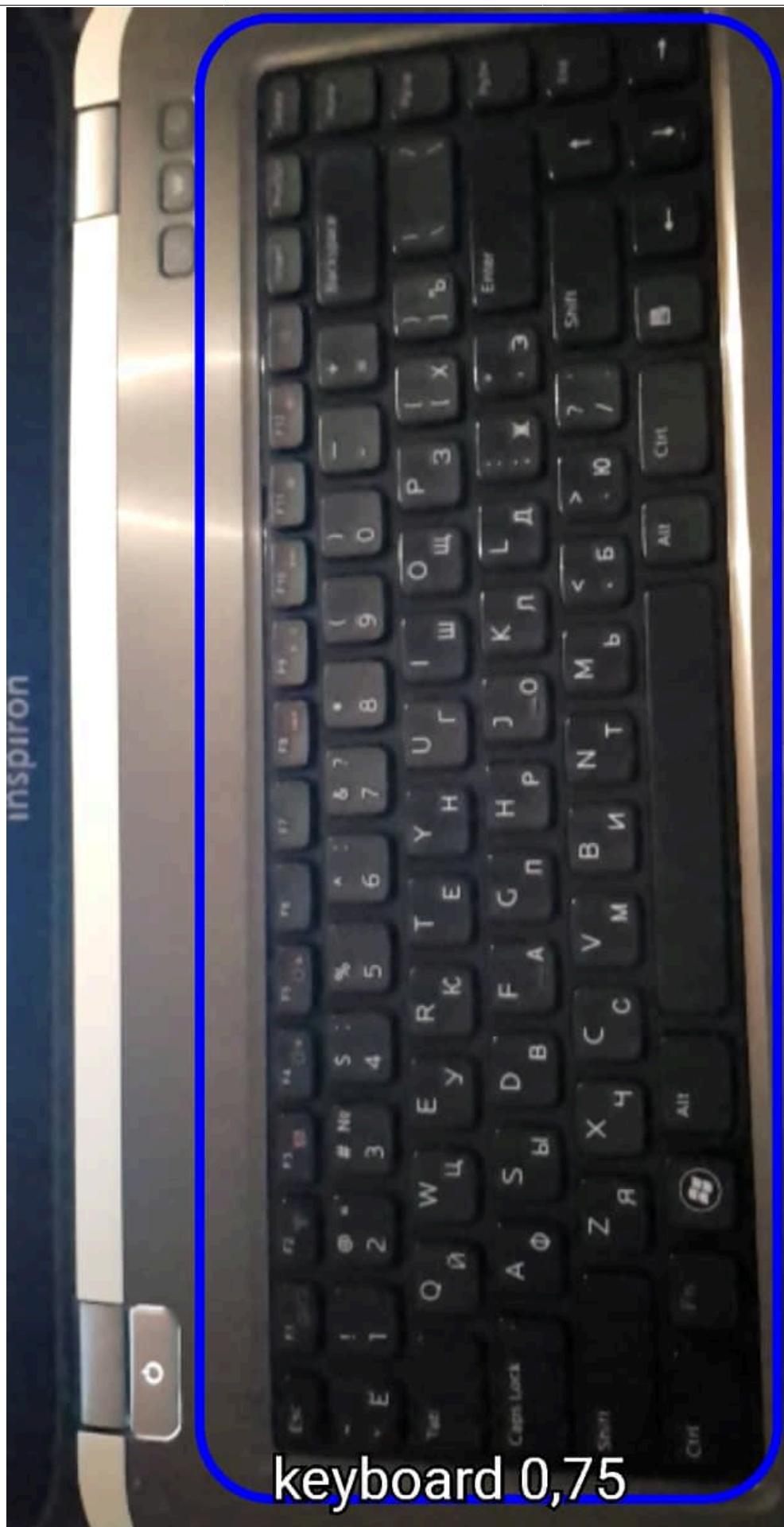




remote 0,97

---

Magas szintű programozási  
nyelvek 2. Jegyzőkönyv.  
Kovács Attila Patrik.



## 0. hét

### C++ és Java

Objektumorientáltság a Javaban és a C++-ban

A Java egy teljesen objektumorientált programnyelv, ellenben a C++-al. C++-ban tudunk procedurálisan is kódolni.

Az objektum egy olyan modellezés, ami a valódi világ valamelyen elemét hivatott leírni/magába foglalni. Magába foglal bizonyos tulajdonságokat és viselkedéseket. Tehát egyszerűen megfogalmazva a Javaban a problémákat minden objektumokkal szeretnénk megoldani.

Az objektumunk ugyanmond tulajdonságai a változói, a viselkedései pedig a metódusai. Az objektum változóit példányváltozóknak nevezzük. A metódusai függvényre hasonlának, valamelyen feladatot végezhetnek el és értékkel térhetnek vissza.

Az osztályok arra szolgálnak, hogy hogy leíjanak egy modellt, ha úgy tetszik sémát, vagy keretet, amikből majd a példányositott objektumok készülnek. Magyarán megmondva leírunk egy osztályt, benne a változókkal és metódusokkal, és a keletkező objektumunk, amikor azt példányosítjuk, ebből az osztályból jön létre.

Minden osztálynak van konstruktora. A konstruktur egy olyan metódus, ami az osztály példányosításakor minden lefut. Ezt az adott osztály írása megírhatja az osztályon belül, de ha nem tenné, akkor a Java és a C++ is generál egy alap konstruktort (tehát minden osztálynak van konstruktora, csak ha azt nem az osztály leíró programozó hozza létre, akkor nem látjuk a kódban). Amikor egy osztályt példányosítunk, és objektum lesz belőle, akkor automatikusan meghívódik a konstruktora, ami felépíti az osztályt. Ha írtunk saját konstruktort az osztályon belül, akkor ezt meg tudjuk hívni paraméterekkel ellátva is, amiket az adott konstruktur fogad.

Az osztályon belül a változóink elérését (módosítását, törlését stb.) nem feltétlenül kell engedélyeznünk mások számára. Az ilyen változókat nevezzük privát változóknak. Mikor egy változó privát, akkor csak az adott objektum tud házzáférni (amelyik az osztályból lett példányosítva). Ahhoz, hogy az ilyen változókhöz hozzáférjenek más objektumok is, az adott objektum metódusait kell segítségül hívni (getter, setter). Tehát az objektumorientált programozás egyik alapjának számít az, hogy egy objektum változóihoz csak felügyelt körülmenyek között tudjon egy másik objektum hozzáérni.

Ezen felül fontos része még az objektumorientált nyelveknek az öröklődés. Vegyük példának egy valódi villági általános dolgot:

Talán a lehető legtipikusabb a madár-pingvin példa. Létezik egy madár osztályunk és egy pingvin osztályunk. A pingvin is egy madár lényegében, tehát minden pingvinről elmondhatjuk, hogy madár is egyben. A madarak sok tulajdonságát és viselkedését már leírtuk a madár osztályában, és amikor írjuk a pingvin osztályt, akkor nem szeretnénk az ugyanolyan tulajdonságokat újra leírni. Erre van kitalálva az öröklődés. Tehát mikor írjuk a pingvin osztályt, akkor nem fogjuk a sok, már megírt tulajdonságot és viselkedést újra leírni, hanem egyszerűen azt mondjuk, hogy a pingvin osztály kiterjeszti (extends) a madár osztályt. Ezzel megörököl minden a madártól. Tudunk a pingvinben még új dolgokat is hozzácsapni az osztályhoz, vagy éppenséggel felülírni bizonyos tulajdonságokat és viselkedéseket, amik a pingvinnél másképp kell kinézzenek, mint a madárnál. Ebben a példában lehet mondani, hogy a pingvin szülője a madár.

Egy talán fontosabb különbség most kerül reflektorfénybe C++ és Java között. Mivel a C++-ban van egy olyan lehetőségünk is, hogy többszörös öröklés (Javaban is létezik ilyen, de csak interfacek esetén, amikről később beszélünk). Tehát a C++-ban egy adott osztálynak több közvetlen szülője is lehet, több osztályt is kiterjeszthet.

Változókról beszélve három féle képpen tudjuk létrehozni őket: public, protected és private módon. Ez a következőket jelenti:

---

Public: amikor egy változót így deklarálunk az adott osztályban, akkor ahhoz a változóhoz mindenki más is hozzáférhet;

Protected: ha így van deklarálva a változó, azt azt jelenti, hogy azon osztályok objektumai, amelyek kiterjesztek az adott osztályt, hozzáférhetnek a változóhoz, de mások nem;

Private: amikor csak az adott osztályból létrejött objektum kap hozzáférést a változóhoz.

Most beszéljünk kicsit az absztrakt osztályokról. Ezek olyan osztályok, amely nem arra lettek szánva, hogy objektumok készüljenek belőlünk. Tehát egy abstract osztályt nem lehet példányosítani. Az ilyen osztályokat egy másik osztály ki tudja terjeszteni. Absztrakt osztályokban általában a metódusokat csak létrehozzuk, de a törzsüket nem dolgozzuk ki, ezt majd megtesszi az őt kiterjesztő osztály. Példának felhozva gondoljunk két osztályra: Állat osztály és Kutyá osztály. Az Állat osztály jelen példában absztrakt kell legyen, mert külön példányosítani senki nem fog olyat, hogy Állat(majd a Kutyát, ami szintén ugye egy Állat). Ebben az osztályban létrehozunk egy olyan metódust, amiből az Állat majd hangot ad ki, mert feltételezzük, hogy minden állat ad ki hangot. Na most; a Kutyá osztály kiterjeszti az Állat absztrakt osztályt, ezáltal tudja magától(az Állatban megírt hangadaás miatt), hogy neki hangot kell tudni adnia. De ugyanezt az absztrakt osztály később kiterjeszheti pld. a Macska osztály is, ami szintén kell hangot adjon, csak eltérőt a Kutyától, ezért is nem írtuk meg az Állatban, hogy mit csinál a "hangadás" metódus, de attól még ott van, és lehet használni. Erre jók tehát az absztrakt osztályok.

Következzenek az interfacek: ez nagyon hasonlít az absztrakt osztályhoz, de van egy nagy különbség kettőjük között. Az interfaceben lévő metódusok CSAKIS absztraktak lehetnek. Tehát az interfacen belül leírhatunk akárhány metódust, de nem lehet törzsük. Ezeket a metódusokat majd az interfacet kiterjesztett osztályban fogjuk felülírni(overrideolni). Itt jön képbe Java esetén a többszörös öröklés: egy adott osztály interfaceból akárhányat kiterjeszthet. Érdekesség még az interfacekkel kapcsolatban, hogy ha egy osztály kiterjeszt egy interfacet, akkor az osztályban az interface minden metódusát felül kell írni.

A metódusok a Javaban és C++-ban is feladatakat hajtanak végre, és visszaadnak értéket annak, ami meghívta őket. Az is lehetségek, hogy egy metódus értéket nem ad vissza, csak végrehajt valamit. A két nyelvben a különbség abban van, hogy Javaban minden egyet metódusnak is egy bizonyos osztály részének kell lennie, míg C++-ban nem.

A memóriaszemét gyűjtését a Javaban egy bizonyos, automatikus működésű ún. garbage collector végzi, ami általában jól végzi a dolgát. C++ esetében viszont nekünk kell gondoskodnunk a felgyült memóriaszemetről.

Kivételkezelés tekintetében minden programnyelv esetében van lehetőségünk rá. Egy ún. try-catch be foglalva az adottakat. Röviden annyi az egész, hogy a try-ban van az, amiben a hiba felütheti a fejét, és a catch-ben kapódik el a kivétel, ha a hiba tényleg létrejön.

## Python

A python egy általános célú, magas szintű programozási nyelv. Alkotója egy Guido van Rossum holland programozó. 1989-ben kezdte el fejleszteni a pythont és 1991-ben hozta nyilvánosságra.

A python támogatja a procedurális és objektumorientált programozást is.

Érdekesség még, hogy a python ún. interpreteres nyelv. Ez azt jelenti, hogy nincs különválasztva a forrás- és tárgykód, tehát a megírt program azonnal futtatható.

Maga a nyelv könnyen megtanulható, egyszerű. Hamar lehet szép, látványos eredményeket elérni vele.

Pythonban egy adott kód általában rövidebb, mint C++-ban vagy Javaban. Ennek az egyszerűsége az oka. Pythonban viszonylag összetett kifejezéseket tudunk írni rövid állításokban és pl. változók definiálására sincs külön szükségünk. Sokak számára még idegesítő és szokatlak ezzel a nyelvvel kapcsolatban, hogy nem használunk sem zárójeleket, sem pontosvesszőket. A kód csoportosítása a sorokkal és tabulátorokkal történik. Az utasítások a sorok végéig tartanak, egy programblokk végét egy kisebb behúzású sor jelzi.

Kommentelni hashtaggel (azaz kettőskereszttel) tudunk.

Python esetében is létezik egy garbage collector a memóriászemét gyűjtésére. Ezen nyelv esetén a változók az objektumokra mutató referenciaiak. A del parancssal töröljük egy változó hozzárendelését, és ha egy objektumra nem mutat egyetlen változó sem, akkor az automatikus garbage collector törlíti azt.

Fentebb említettem, hogy a változókat nem kell külön deklarálni. Ez azt is jelenti, hogy a típusukat sem kell leírnunk, a python kitalálja magától. Az adattípusok lehetnek logikaiak(true/false), számok, szövegek(string), listák, halmazok és szótárak(kulcs-érték párok). Érdekesség, hogy a számok esetén a szám lehet komplex szám is.

Alapvetően a változóink mindenhol helyi, azaz lokális változók, ha ezen változtatni szeretnénk (hogy globálissá tegyük), akkor ezt a global parancssal tudjuk megtenni. A változók közötti konverzió is támogatott, tehát tudunk stringet intté tenni pl., de akár intet is floattá, ha lehetségek adott körülmények között.

Kiiratni a print parancssal tudunk. Több dolgot zárójelein belüli vesszővel történő elválasztással.

Ciklusok tekintetében megtalálható a for is, illetve a while is. Bennük ugyanúgy létezik a break és a continue.

Labelek is vannak, amiket ha elhelyezünk valahová, akkor oda a goto parancssal tudunk ugrani.

A függvényeket a def parancssal tudjuk definálni. A függvények, mintha értékek lennének, továbbadhatók más függvényeknek, illetve objektumkonstruktornak is. Ugyanúgy, mint más nyelvek esetén, itt is lehetnek a függvényeknek paramétereik. A legtöbb paraméter érték szerint adódik át, kivéve a mutable típusokat. A függvényeknek egy visszatérési értékük van.

Objektumok tekintetében hasonlít a C++-hoz, illetve a Java-hoz. Írhatunk osztályokat is, amiket majd példányosíthatunk. Ezek tartalmazhatnak metódusokat, amiket akár más osztályoktól is örökölhetnek. Ezekben felül az osztályoknak lehet egy speciális, konstruktur tulajdonságú metódusa, az \_\_init\_\_.

Amiről még érdemes szót ejteni, az a kivételkezelés, ami itt is létezik. Parancsa a try és except, és lehet else ág is. A tryban a kód, amit meg akarunk próbálni, amelyben a kivétel előállhat. Ha hiba lép fel, akkor az except blokkra ugrik, és az ebben lévő utasítások hajtódnak végre.