

Varga Márton

INBPM0315-17 Magas szintű programozási nyelvek 2*

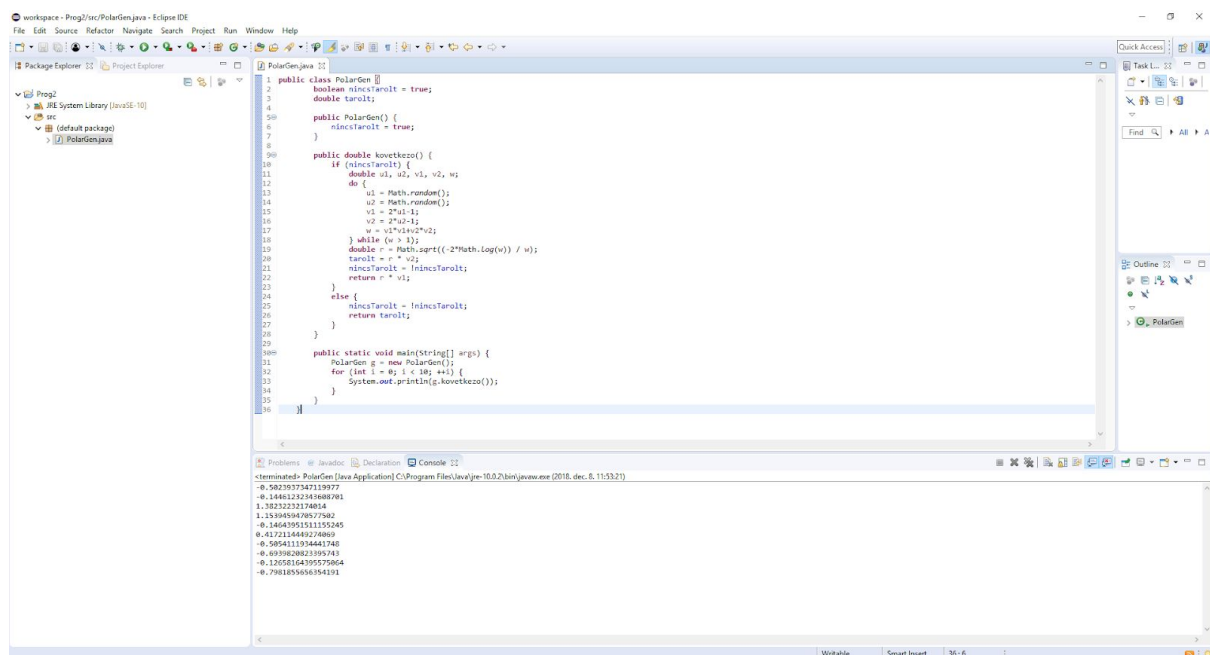
1.hét

OO szemlélet :

Az Objektorientált programozás egy paradigma ami a megoldandó problémák objektum-orientált módon való megközelítése.

Ebben feladatban létrehozzunk egy PolarGen nevű osztály, amit a main-ben példányosítunk. A példányosítás során nincsTarolt nevű adattag true lesz ezért az if vezérlésátadás végrehajtódik és addig hajtja végre a do while –on belüli műveleteket, ameddig a w kisebb nem lesz mint 1. Ezt követően a program vissza tér r*1v1 értékkel. Ha nincsTarolt nevű adattag hamis akkor a program a tárolt értékkel tér vissza.

Java:

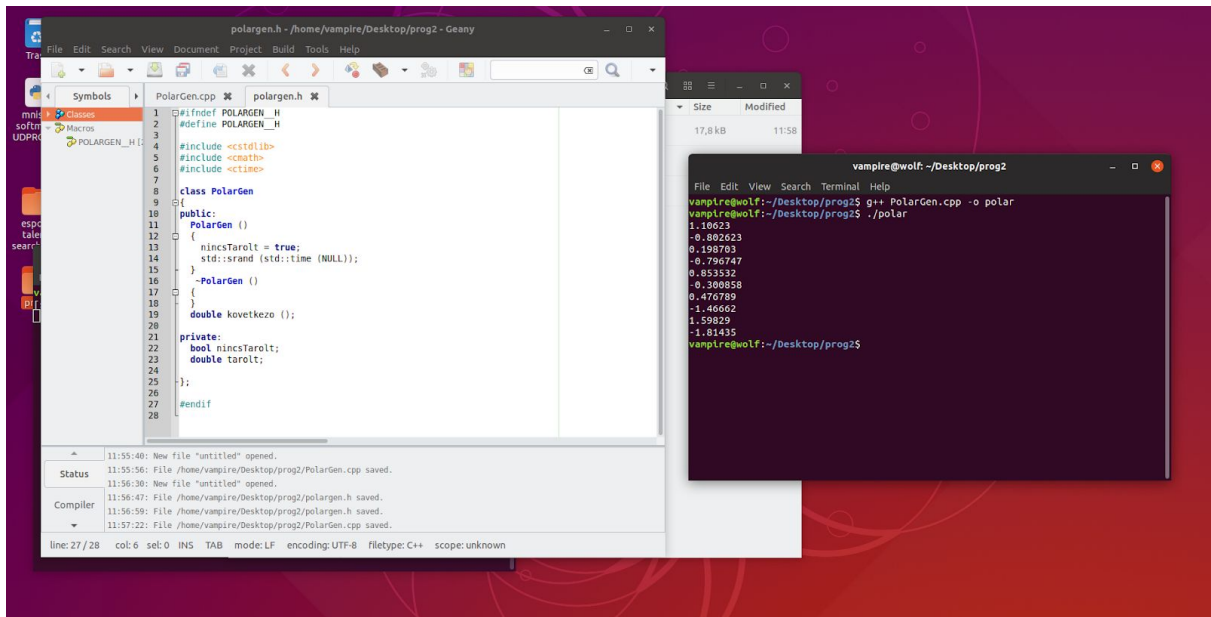


```
1 public class PolarGen {
2     boolean nincsTarolt = true;
3     double tarolt;
4
5     public PolarGen() {
6         nincsTarolt = true;
7     }
8
9     public double kovetkezo() {
10        if (nincsTarolt) {
11            double u1, u2, v1, v2, w;
12            do {
13                u1 = Math.random();
14                u2 = Math.random();
15                v1 = 2*u1-1;
16                v2 = 2*u2-1;
17                w = v1*v1+v2*v2;
18            } while (w > 1);
19            double r = Math.sqrt((-2*Math.log(w)) / w);
20            tarolt = r * u2;
21            nincsTarolt = !nincsTarolt;
22            return r * v1;
23        }
24        else {
25            nincsTarolt = !nincsTarolt;
26            return tarolt;
27        }
28    }
29
30    public static void main(String[] args) {
31        PolarGen g = new PolarGen();
32        for (int i = 0; i < 10; ++i) {
33            System.out.println(g.kovetkezo());
34        }
35    }
36 }
```

Console Output:

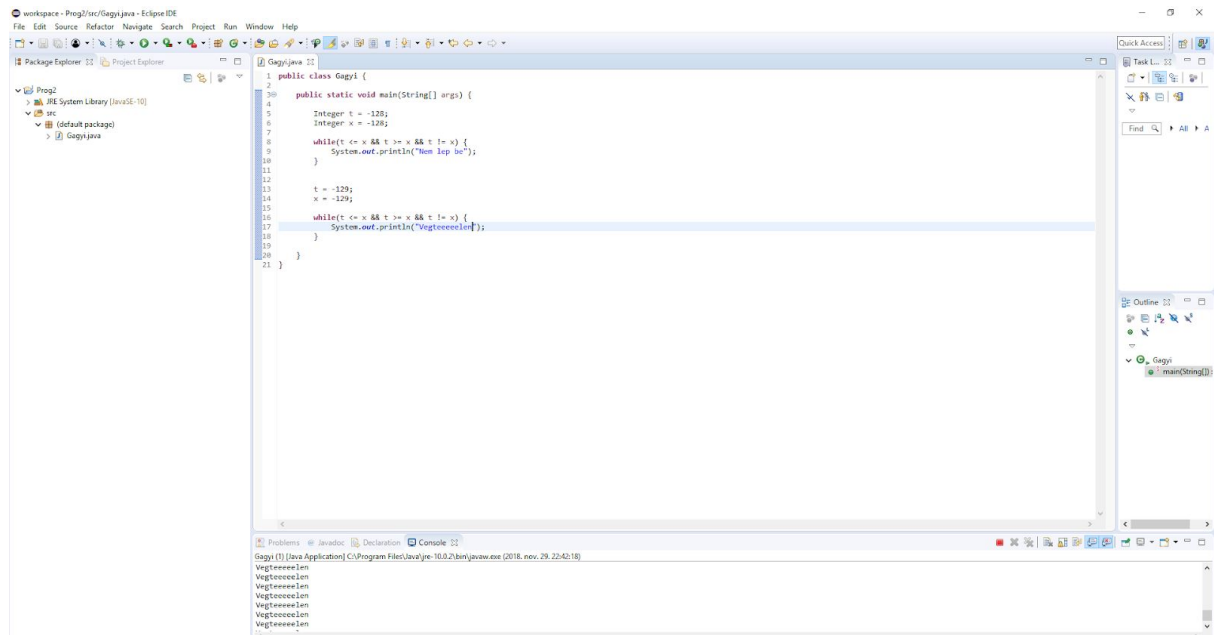
```
<terminated> PolarGen [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2018. dec. 6. 11:53:21)
-0.5825937347113977
-0.14861232341868791
1.182323232174014
1.15304068780775402
-0.1466395151155245
0.4172114448274809
-0.5954111934441748
-0.4939820823395743
-0.13050164439575064
-0.798185568354191
```

C++:



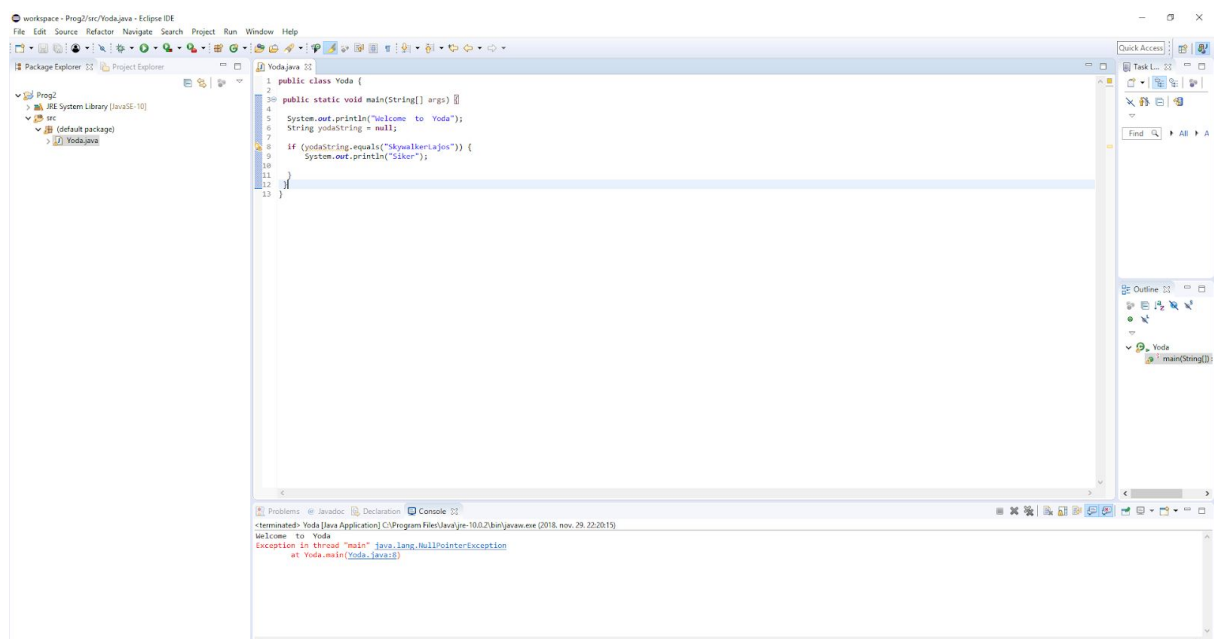
“Gagyi”

Ha gatyi feladatban megadott t és x nevű változó az adott -128 és 127 közötti intervallumon kiesik akkor az érték nincs letárolva és ekkor egy new in(i) értékel tér vissza. Ezért a while ciklusban megadott felétel végtelen lesz azért mert $!=$ operátor igazat fog adni. Ha az intervallumon belülre esik akkor az object előre letárolt értékekkel dolgozik így a végtelen ciklus nem jön létre.



Yoda:

A Yoda feltétel egy programozási konvenció. Ahol `myString.equals("ez")` esetén az if vezérlésátadó utasítás hamisat és `NullPointerException` hibaüzenetet dob vissza így leáll a program. De a yoda feltételt felhasználva pedig a vezérlésátadás ismét hamis lesz de a program nem áll le.



2.hét

Ciklomatikus komplexitás:

A komplexitás kiszámolásához a lizaard.ws nevű weboldalt vettem igénybe.

A ciklomatikus komplexitás azt számolja, hogy a programban hogy egyéni út járható be.

A BBP és a fullscreen egyik eredeti verziójának néztem meg a komplexitását.

The screenshot shows the Lizard code complexity analyzer interface. The header includes the site name and navigation links. The main content area has a heading "Complex is better" and a brief description of the tool. Below this, there are two panels. The left panel, titled "Try Lizard in Your Browser", shows a code editor with a Java Swing window titled "Fullscreen". The right panel, titled "Code analyzed successfully.", displays the analysis results in a table.

File Type	Java	Token Count	115	NLOC	16
Function Name	NLOC	Complexity	Token #	Parameter #	
Fullscreen: main	12		1	94	

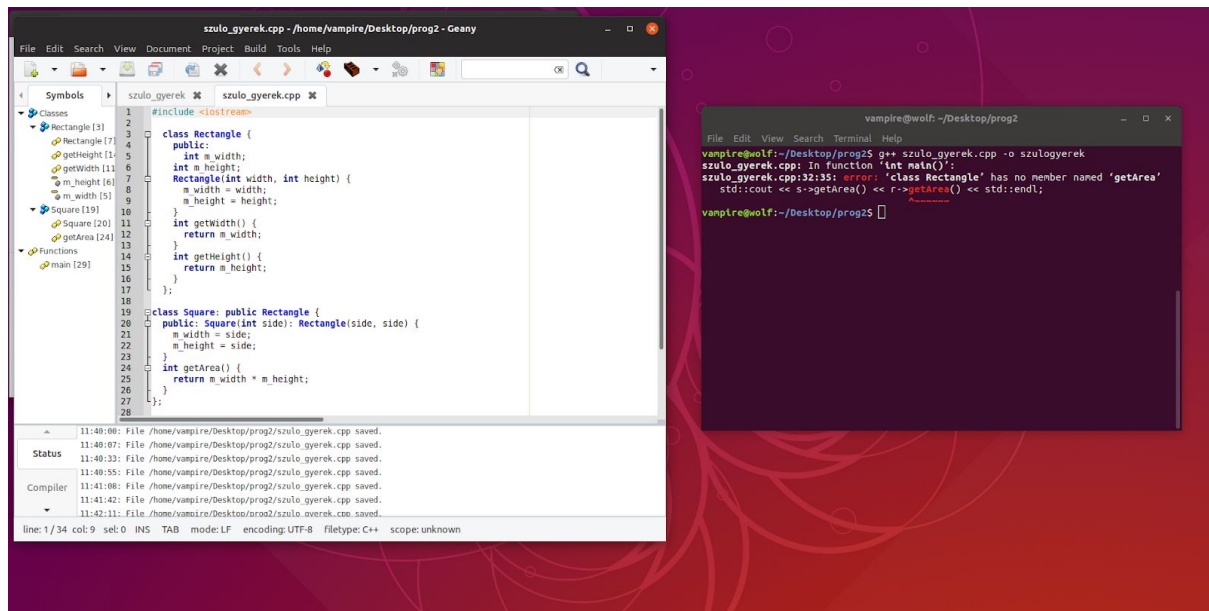
The screenshot shows the Lizard code complexity analyzer interface. The header includes the site name and navigation links. The main content area has a heading "Complex is better" and a brief description of the tool. Below this, there are two panels. The left panel, titled "Try Lizard in Your Browser", shows a code editor with a Java program titled "BBP". The right panel, titled "Code analyzed successfully.", displays the analysis results in a table.

File Type	Java	Token Count	450	NLOC	65
Function Name	NLOC	Complexity	Token #	Parameter #	
BBP: BBP	22		3	187	
BBP: mod	22		5	101	
BBP: sum	10		3	125	
BBP: toString	3		1	8	
BBP: main	3		1	22	

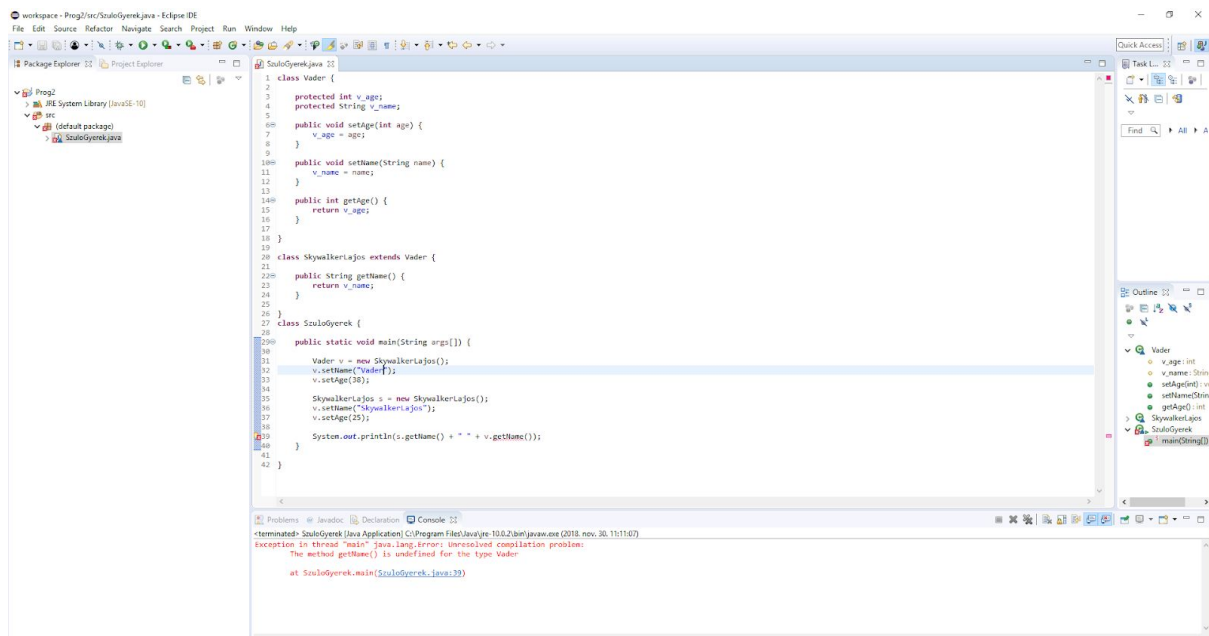
Szülő-gyerek:

A feladat azt mutatja be, hogy szülőn keresztül nem lehet alosztály függvényét meghívni hiába a szülő része az alosztály. Ha például egy gyerek objektumban hoznánk létre egy egy szülő osztályt akkor sem lehetne hozzáférni A feladat bemutatja a szülő-gyerek objektum kapcsolatát.

C++:



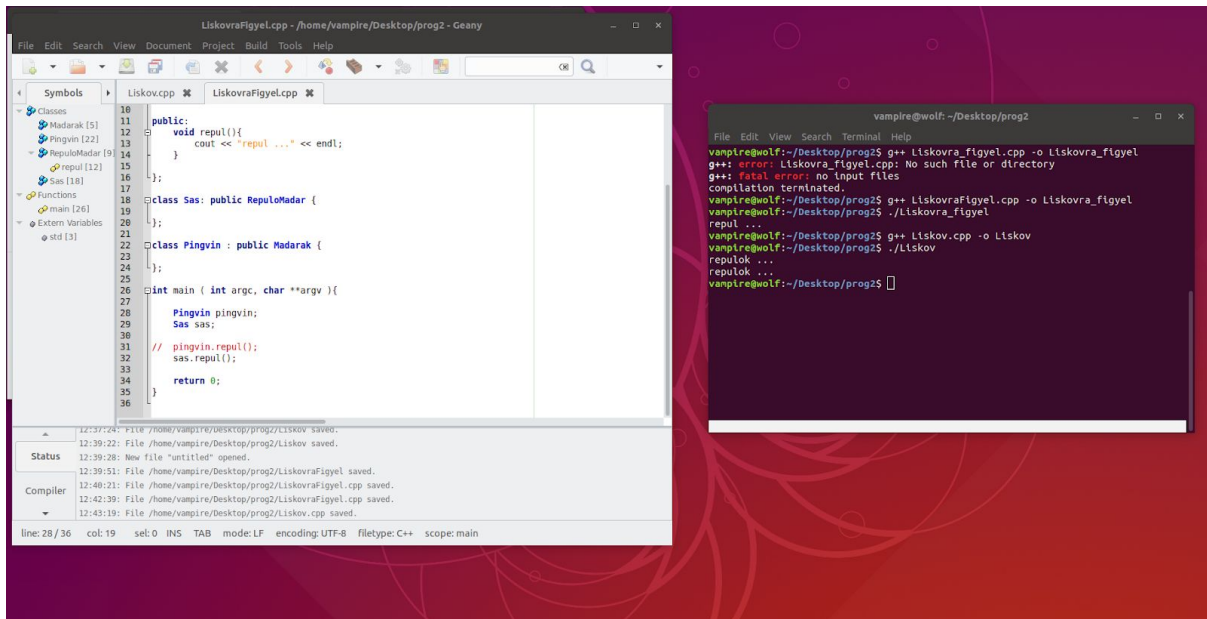
Java:



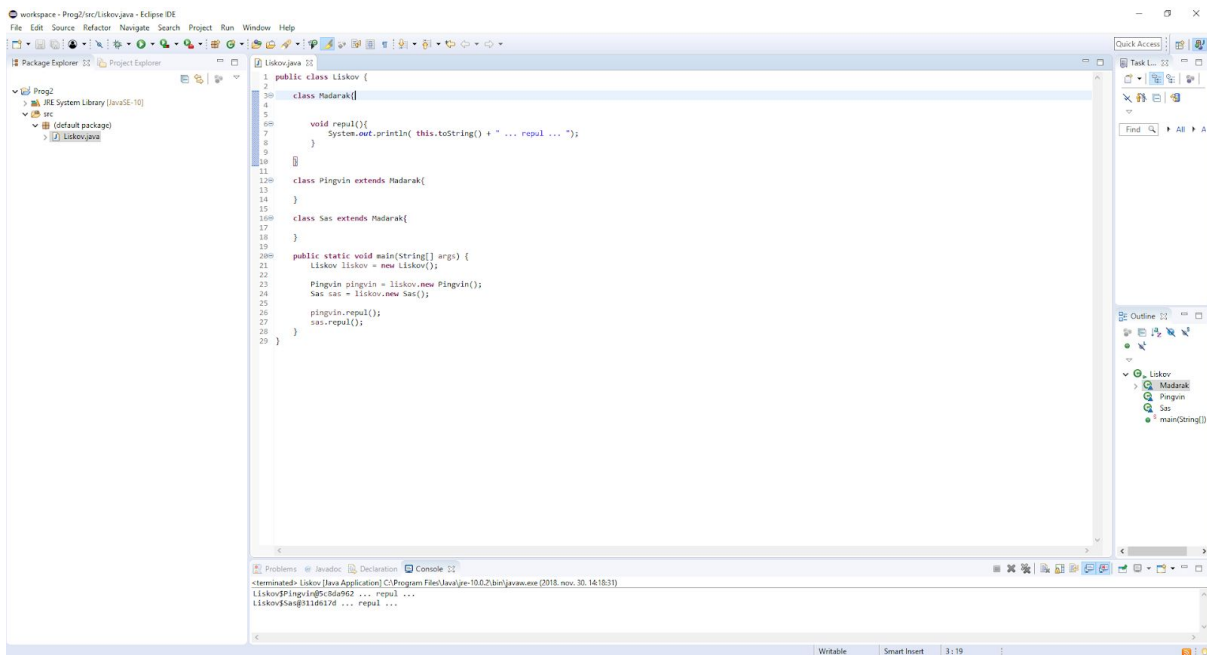
Liskov:

A Liskov elv szerint az osztályokat úgy kell megtervezni: ha M altípusa N-nek, akkor minden olyan helyen ahol N-t felhasználjuk M-t is minden gond használhatjuk anélkül hogy a programrész tulajdonságai megváltoznának. Az első képen a C++ verziót szemlélteti a másik kettő pedig a javat hivatott.

C++:



Java:



```

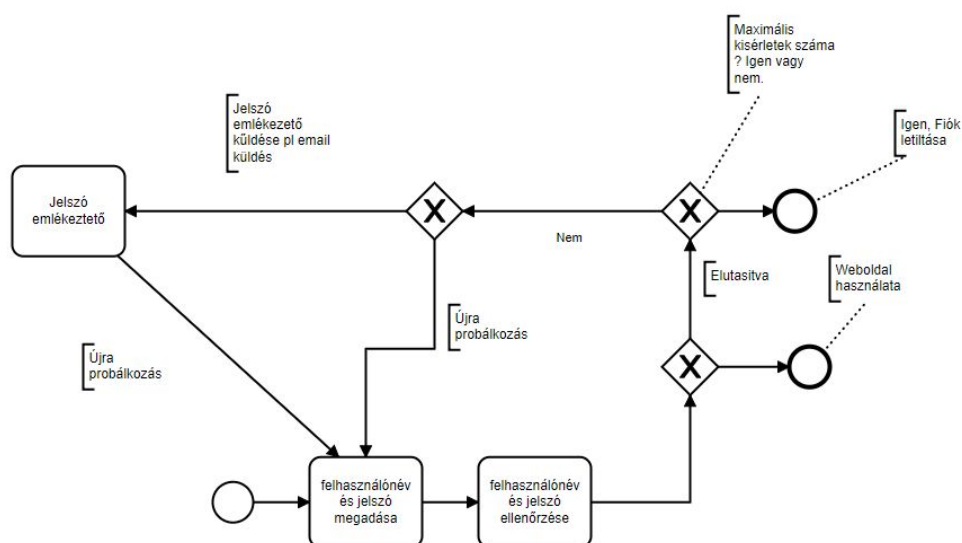
1 public class Liskova_figyel {
2     class Madarak {
3     }
4     class RepuloMadar extends Madarak {
5     }
6     void repul() {
7         System.out.println( this.toString() + " ... repul ... ");
8     }
9 }
10 class Pinguin extends Madarak {
11 }
12 class Sas extends RepuloMadar {
13 }
14 public static void main(String[] args) {
15     Liskova_figyel liskov = new Liskova_figyel();
16     Pinguin pinguin = liskov.new Pinguin();
17     Sas sas = liskov.new Sas();
18     // pinguin.repul();
19     sas.repul();
20 }
21 }

```

3.hét

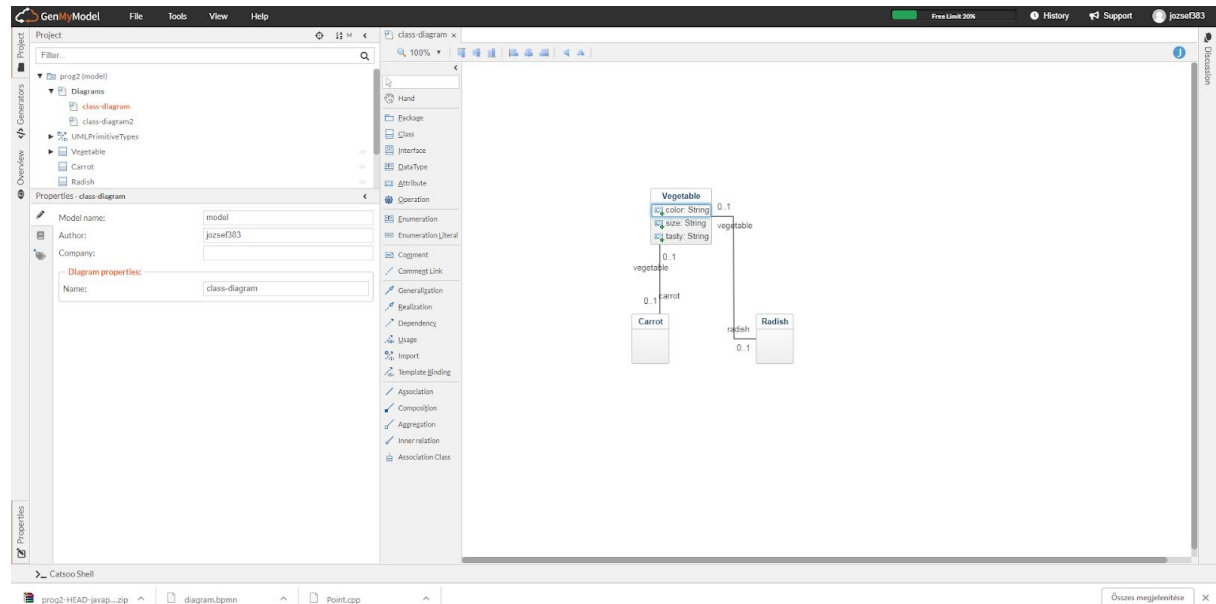
BPMN:

BPMN avagy Business Process Model and Notation egy grafikus megjelenítése az üzleti folyamatoknak. A feladat megoldása során a bpmn.io nevű weboldalt vettem igénybe. A saját BPMN folyamatábrámon egy internetes felhasználófiókba történő bejelentkezés folyamata szerepel.



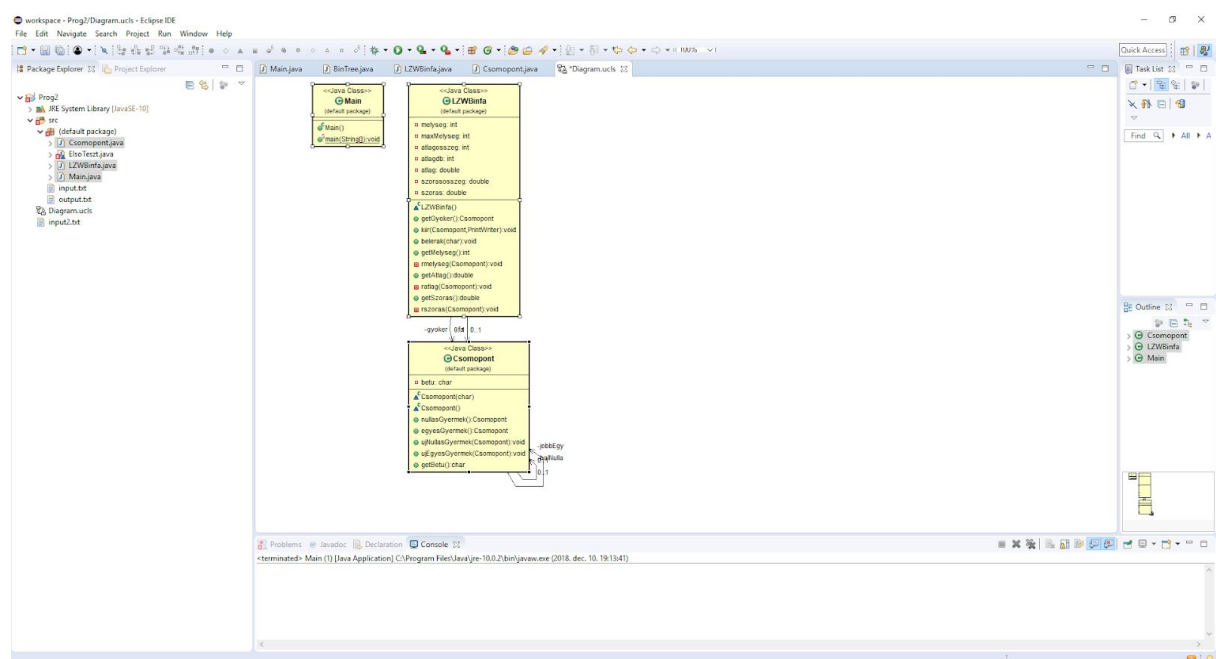
Forward engineering UML osztálydiagram :

Itt egy egyszerű öröklődési uml generáltam osztályokat.



Reverse engineering UML osztálydiagram

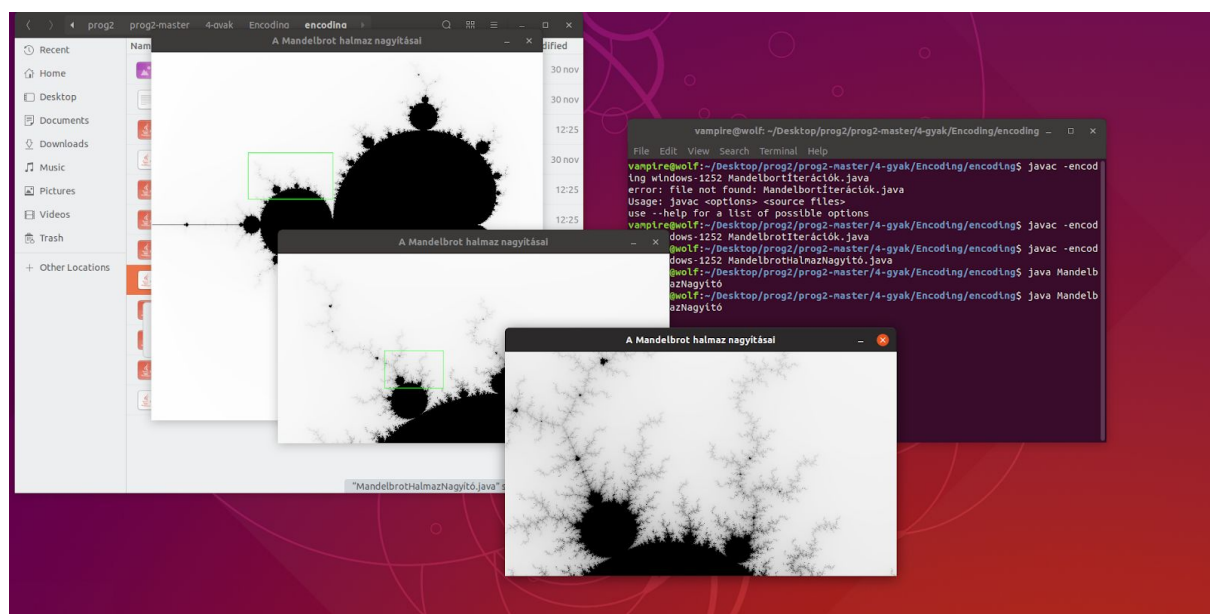
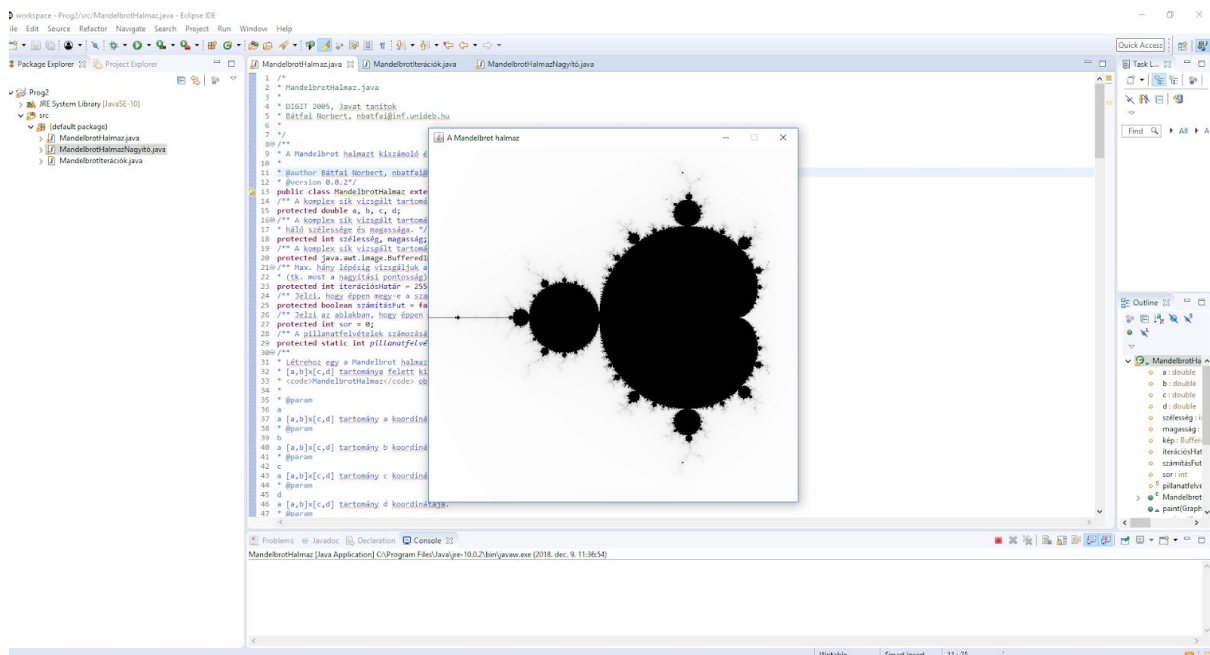
Az Uml avagy Unified Modeling Language egy objektum orientált modellező megoldás nagy méretű programok dokumentálására. Ehhez a feladathoz objectaid.com nevű eclipse plugint vettem igénybe. Az LZWBinFa sikeres importálása után a program diagrammot generált belőle.



4.hét

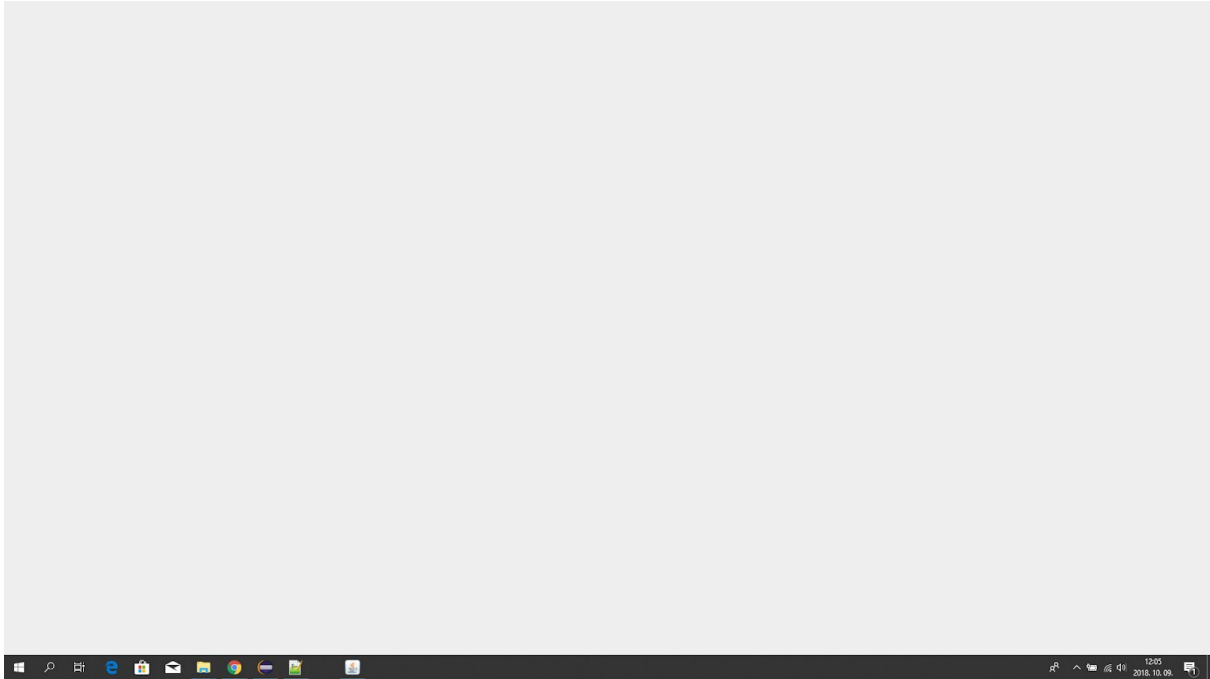
Encoding :

Az adott e heti feladatban a megkapott link letöltése után egyszerűen futtatni kell a MandelbrotHalmazNagyító.java nevű fájlt. A fordítás MandelbrotHalmazNagyító.java -val az ubuntu-ban történik ugyanis ott sikerült rá közelítenem. Futtatni pedig java MandelbrotNagyító -val. A MandelbrotHalmazNagyítóban sikeresen rá tudunk közelíteni a mandelbrot halmazra.



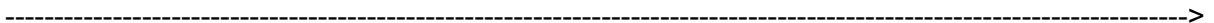
Full screen:

Egy olyan algoritmust kellett készíteni, ami teljes képernyőt befedi.



l334d1c48:

A **l334d1c48** leetspeak a világhálón létrejött jelrendszer ami szokásos vagy hétköznapi karakterek helyett más karaktereket használ. Az emberek legfőképp az egyszerűbb változatát részesítik előnyben pl (e->3;t->7 stb.). A program algoritmusá beolvas egy fájlt azaz hogy mit mire fordítson majd ezt követően vissza adja a leet változatát.



Mozgató konstruktor: A másoló konstruktor mellé megadhatunk egy mozgató konstruktort is, ami nem tartja meg az eredeti objektum tartalmát a másoló konstruktorral ellentétben, hanem átrakja azt az új objektumba, ezzel memóriát spórolunk.

A Boost rengeteg multiplatform könyvtár halmaza, ami olyan struktúrákat és feladatokat segít, mint a lineáris algebra, random szám generálás, többszálú programozás, metaprogramozás, lambdák, tetszőleges méretű számok stb. Rengeteg munkahelyen használnak Boostot, így elsajátítása jelentősen növelheti az esélyeinket. C++11-es szabvány sok mindent átvett a Boostból (pl.: Smart pointerek, Lambdák.).

A Boost csapat 1998 környékén alakult, amikor a szabvány legelső változata megjelent. Azóta csak folyamatosan nőtt, és most nagy szerepet játszik a C++ szabványosításában. Bár a szabványosítási bizottság és a Boost között nincs formális kapcsolat, a fejlesztők egy jelentős része mindkét csoportban aktív. A 2011-ben jóváhagyott C++ szabvány jelenlegi verziója olyan directorykat tartalmaz, amelyek gyökerei a Boost csapatban találhatók.

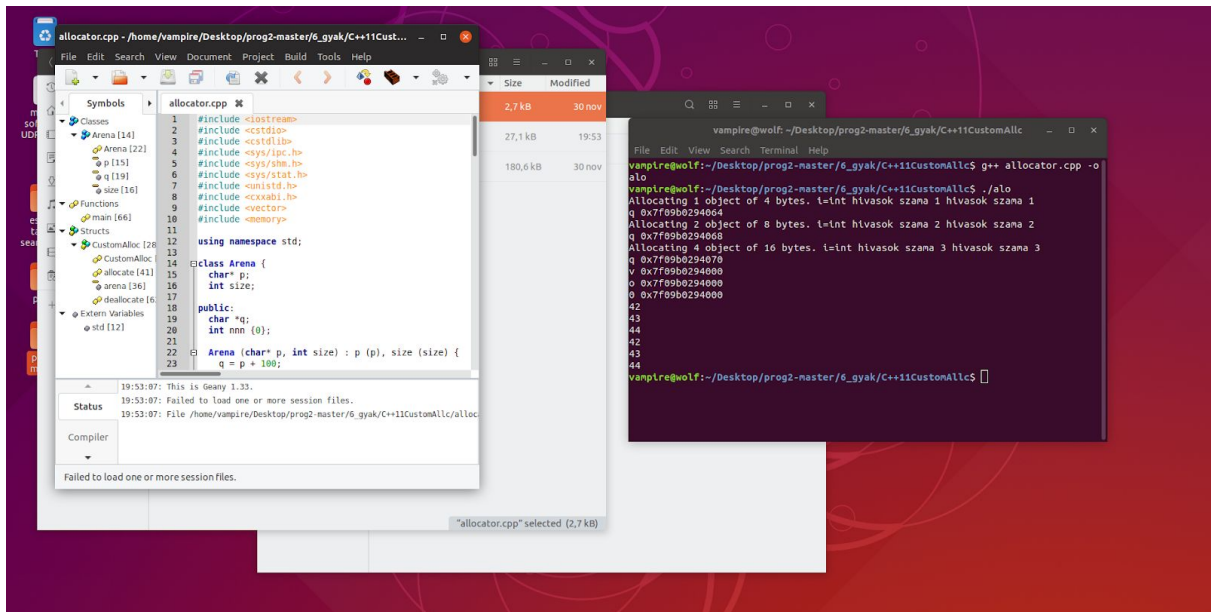
A Boost directoryk választást jelentenek a C++ projektek fejlődésének növelésére, ha a követelmények meghaladják a szabványos könyvtárban rendelkezésre álló jelenlegi adatokat. Mivel a Boost directory gyorsabban fejlődnek, mint az átlagos könyvtár, korábban hozzáférhettek az új fejlesztésekhez, és nem kell várnia, amíg ezeket a fejlesztéseket hozzáadják a szabványos könyvtár új verziójához. Így a Boost könyvtáraknak köszönhetően élvezheti a C++ gyorsabb fejlődését.

----->

6.hét

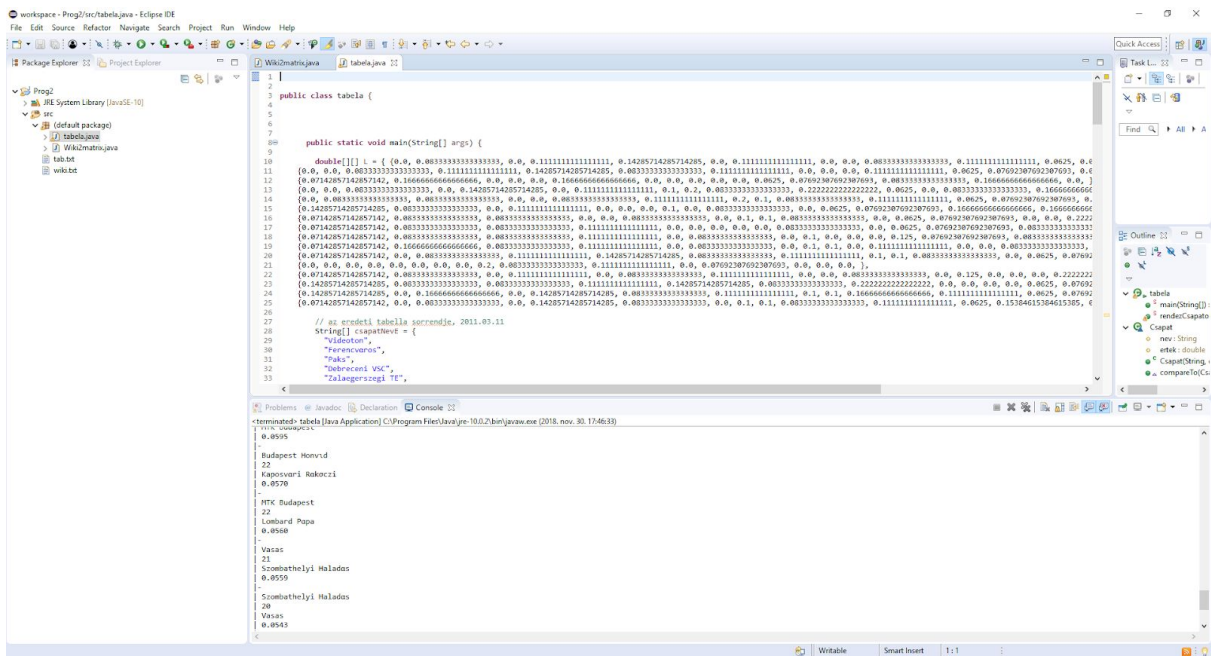
C++11CustomAllocator:

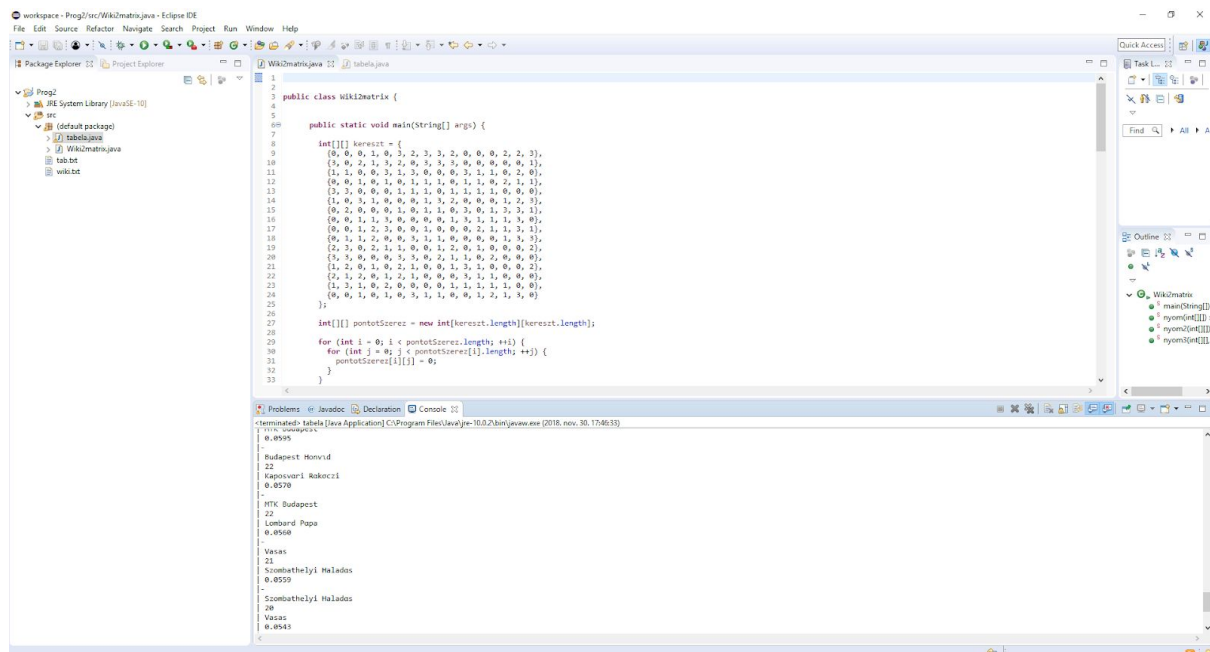
Az allocatorok segítségével tudjuk rendre szabályozni, hogy az algoritmusunk hogyan legyen képes a memória foglalására erre a c++ magától is képes de a minél hatékonyabb programírás érdekében magunk is írhatunk allocatort.



Alternatív Tabella rendezés:

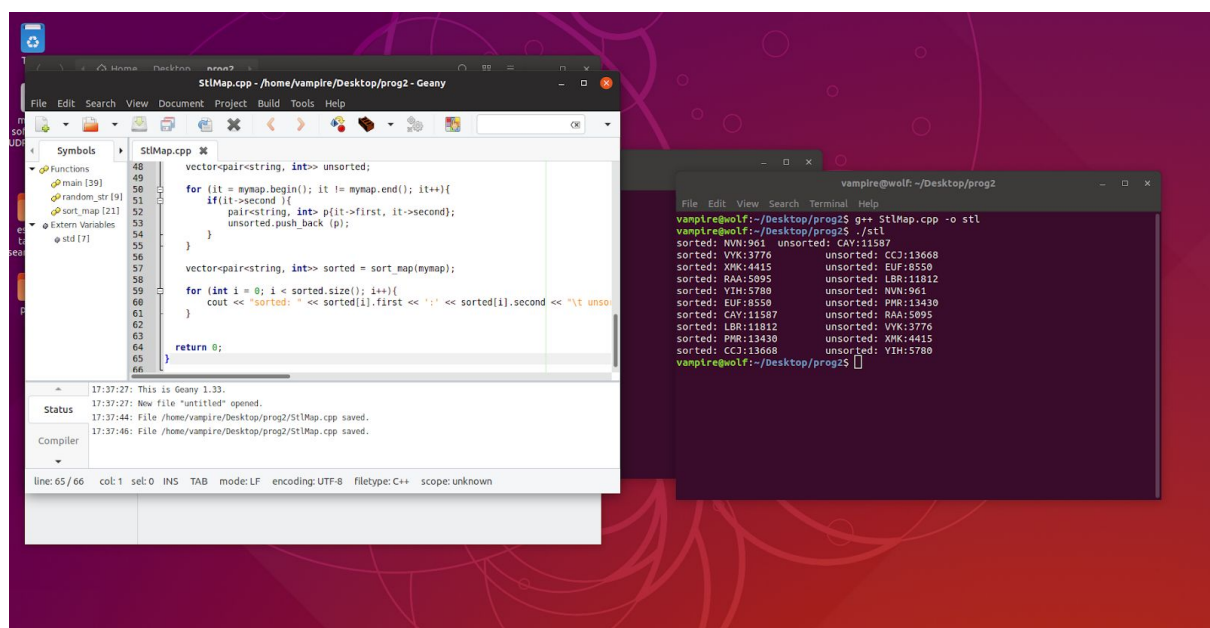
A javában az úgynevezett comperable interface felhasználásával az osztály egyetlen adata alapján is képesek vagyunk rendezni. Maga az interface a `java.lang` package-ben található, és egyetlen metódusa van a `compareTo(Obj o)`. Azt követően, hogy megadtuk a `Comparable` interface-t implementáló `Csapat` osztályunkat, és azt, hogy hogy viselkedjen a `compareTo` metódus, tudjuk rendezni tudjuk a rendezni a kiválasztott osztályt.





STL map érték szerint rendezése:

A `std::map` nem más mint egy úgynevezett rendezett asszociatív tároló, amely egyedi kulcsokkal rendelkező kulcs-érték párokat foglal magába.



Összefoglalás

Standard Template Library avagy a STL c++ szabványos Sablonkönyvtára függvénysablonokat osztály sablonokat tartalmaz így lehetővé téve számunkra, hogy

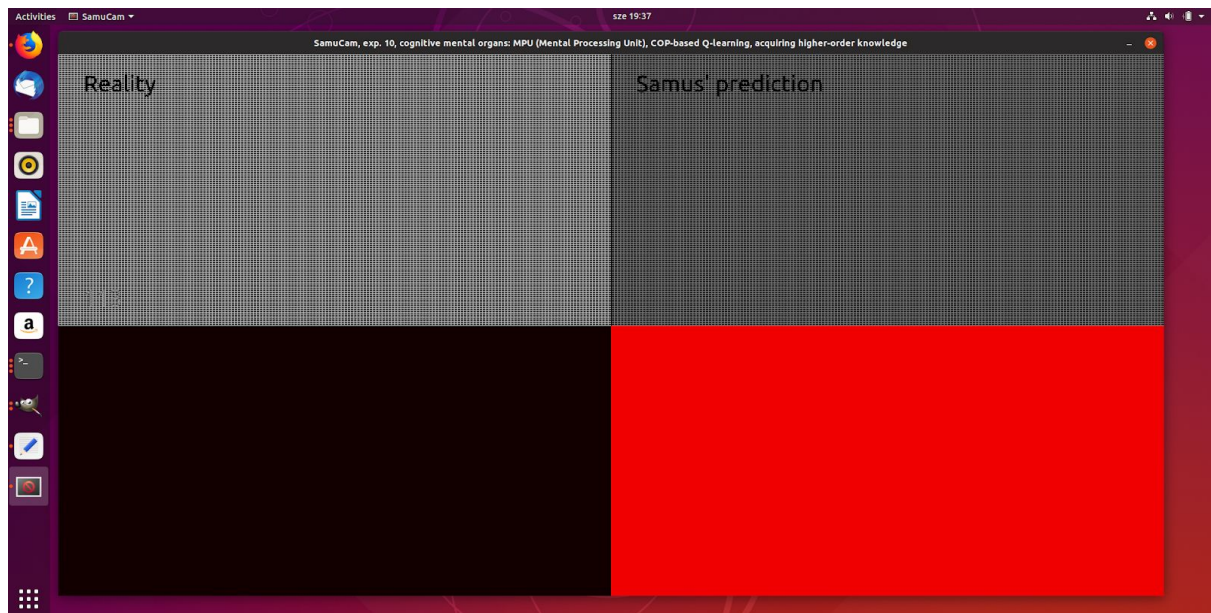
különbéféle algoritmusokat mint pl keresés összefésülés stb és népszerű adatstruktúrákat úgy mint a halmaz, szótár, vektor stb rakhatunk be a kódunkba. A sablonos megvalósítás elérhetővé teszi azt a fejlesztőknek hogy egy adott néven szereplő függvényeket, osztályokat szinte minden tipushoz felhasználhatjuk. Alapvetően három pillére építkezik az STL a literálokra, a konténerekre, és az algoritmusokra. Egyszerűen végig gondolva az algoritmusokat a konténerekben tárolt adatokon hajtjuk végre az iterátorok használatával. Az STL egyik fontos része a tárolók, azok az adatszerkezetek, amelyek különféle eltárolási stratégiákat implementálva biztonságosan, hatékonyan és típus helyesen képesek tárolni az adatokat, nem úgy mint a C-stílusú, kézzel írt láncolt, beépített tömbökkel és adatszerkezetekkel. Az egyedileg létező kulcsot tartalmazó asszociatív tárolók (map,set, stb.) az absztrahálás esetén csak akkor lesz sikeres, ha a kulcs még volt nyilvántartva a konténerben. Az ilyen containerek(tárolók) használatával a ret visszaadott érték egy `pair<iterátor,bool>` pár, ahol az iterátor a beszúrt elemre hivatkozik, vagy arra, ami megakadályozta a beillesztést, a logikai érték pedig `true`-val jelzi, ha megtörtént a művelet. A több, ugyanolyan kulcsot is megengedő asszociatív tárolóknál (multiset, multimap, stb.) a ret egy iterátor, ami kijelöli a beszúrt elemet.

7.hét

SamuCam:

Ehez a programhoz Opencv3.1.0-t, és Qt5.11.2-t használtam.

A SamuCam bepöccítéséhez a .pro file-t módosítottam: hozzáadtam a LIBS-hez -lopencv_core-t és sok egyebet.



BrainB:

Ehez a programhoz Opencv-t és Qt-t használtam.

A BrainB elindításához a .pro file-t módosítottam: hozzáadtam a szükséges LIBS-hez -lopencv_core-t pl.

A Qt signalok az a célja, hogy jelezzék, ha az objektum megváltozott.

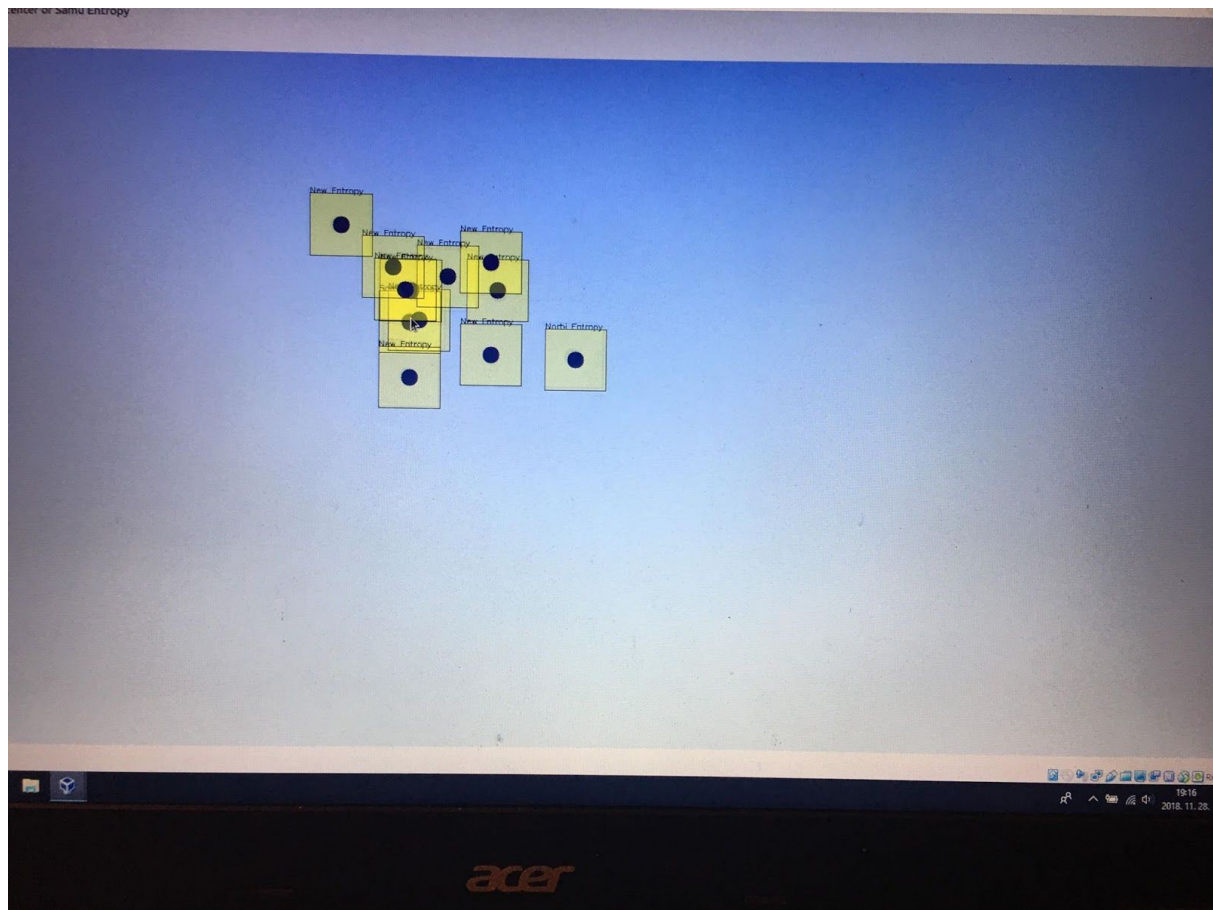
A SLOT-ok egyszerű metódusok amiket meg tudunk hívni a main-en belül is.

A Qt slot signal a BrainB esetén a connect-tel a brainBThread-ben lévő signal hatására végrehajtja az

objektumon a slot utasítást.

(pl. a brainBThread-ben lévő heroesChanged signal hatására ez az objektum végrehajtja a slotnak

kikiáltott updateHeroes funkciót)



Future:

Az ActivityEditor egy javaban íródott program, aminek a futtatásához javafx kell.

A program feladata a FUTURE szerekezetének feltérképezése, vizuális megjelenítése, illetve

szerkesztése.

Az volt a feladat, hogy javítsunk a ActivityEditoron, szóval hozzáadtam egy új menüpontot, amit

jobbgyommbal való kattintással hozhatunk elő.

Az új menüpont: „Elérési út”, amire hogyha rákattintunk, akkor kiírja a terminálra az utat.

Source code:

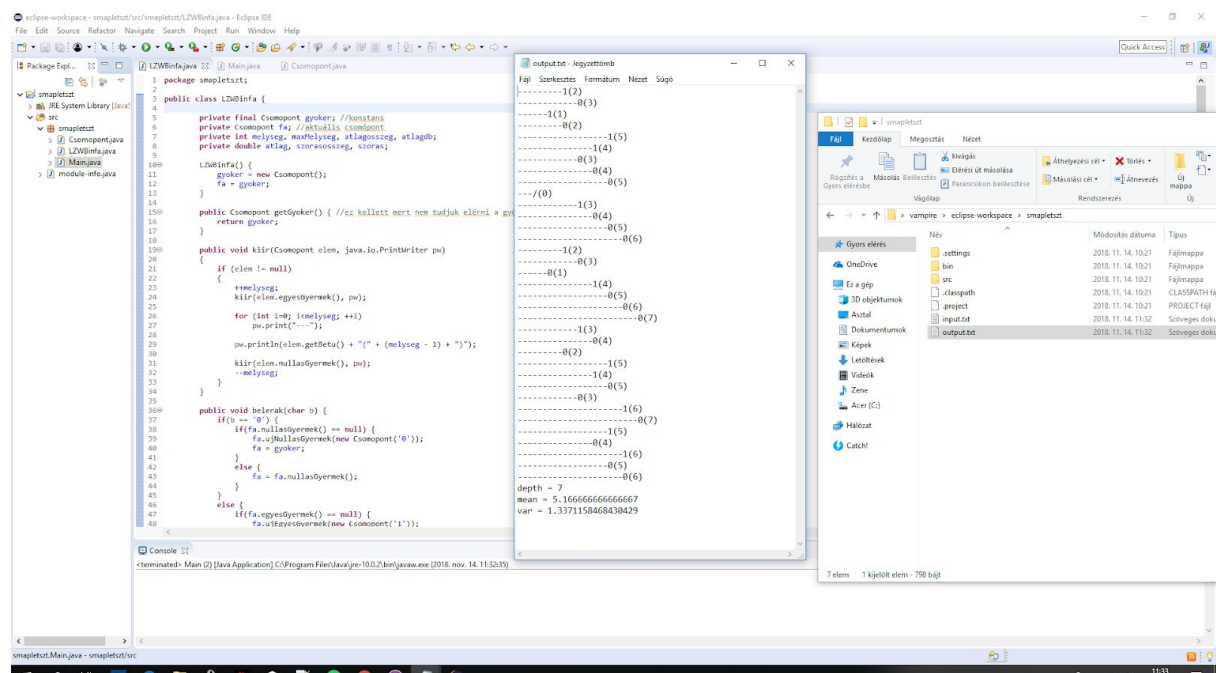
https://github.com/martva383/prog2/tree/master/7_gyak/Future

8.hét

AOP:

Az aspektus orientált programozás az AOP egy programozási gondolkodás, aminek célja hogy a programunkat

kisebb részekre bontsuk, ezzel hatékonyabbá, könnyebben megérthetővé téve azt.



Összefoglaló:

Definíció: A kivétel egy olyan esemény, amely a program végrehajtásakor keletkezik, megszakítva az utasítások végrehajtásának normális folyamatát. Ha egy metódusban hiba lép fel, a metódus egy objektumot generál, melyet átad a futtatási környezetnek. Az objektumot kivétel objektumnak neveznek magába foglalja az információt a hibáról, annak típusáról és program állapotáról, amikor a hiba keletkezett.. Kivételdobásnak azt nevezzük amikor a kivétel

objektum létre jön és a rendszer kezeli. Miután a metódus eldobja a kivételt, a futtató környezet megpróbál a kezelésére találni valamit. A lehetséges dolgok, melyek a kivételt kezelik a meghívott metódusok rendezett listája abban a metódusban, ahol a hiba keletkezett. A metódusok listáját hívási veremnek nevezzük. Ha a futtatókörnyezet a metódusok végig nézése után sem talál neki megfelelő kivételkezelőt, a futtató rendszer és ez által a program) leáll.

Port Scan:

scan:

Ennél a kivételkezelésnél 2 kapcsos zároljelpárt definiáltunk. Ez a két blokk a „try” illetve a „catch” kódblokkok.

Az első blokk az úgymond „megpróbálni” itt helyezzük el azokat az utasításokat, amelyeket szeretnénk végrehajtani, hogy észre vegyünk hogy hibát adnak e vissza. Ebben a

az algoritmusban a ezek a következő utasítások:

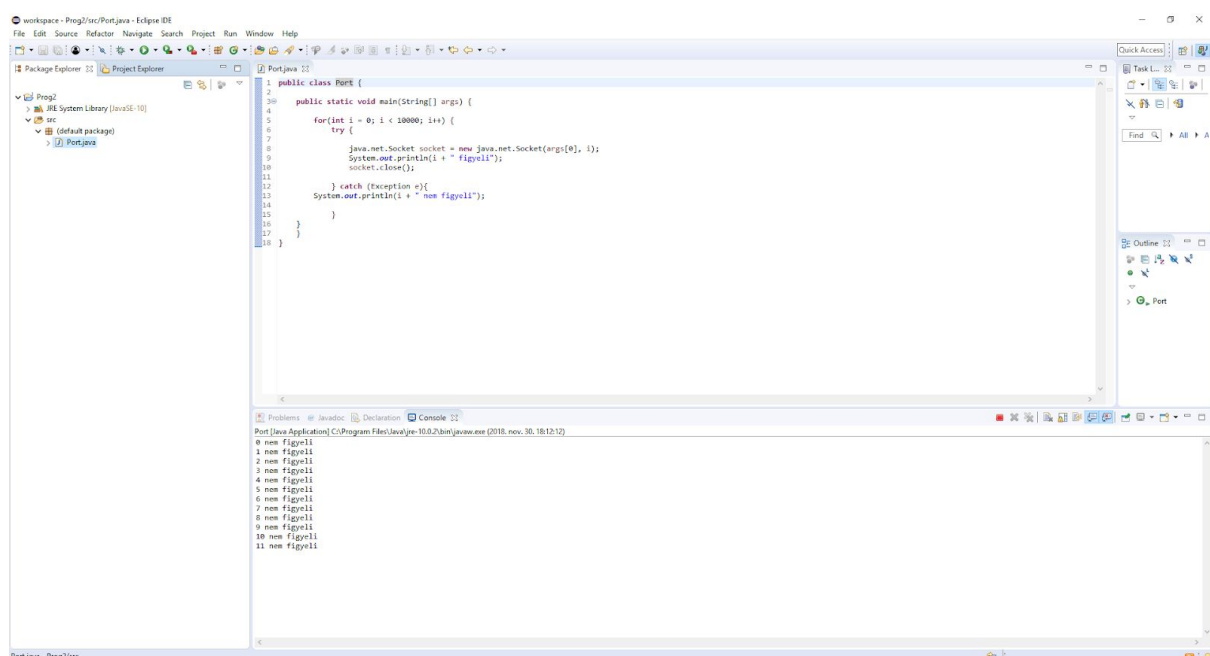
```
java.net.Socket socket = new java.net.Socket(args[0], i);
```

```
System.out.println(i + „ figyeli”);
```

```
socket.close();
```

Ha ezek közül bármelyik is hibát készít (ebben a példában valószínűleg az első sornál fog hibát találni), akkor végrehajtnak azok az utasítások, amelyek a „catch” ágban szerepelnek. Ebben az esetben kiírja a program, hogy nem figyeli az adott portot.

Lényegében a program annyit csinál, hogy végigmegy 0-tól 1023-ig a portokon, és megpróbál mindegyikkel TCP kapcsolatot felvenni, ha ez sikerül, akkor képernyőre írja, hogy az adott portot figyeli, de ha bármilyen utasítás ami a „try” blokkban szerepel generál, akkor a program kiírja, hogy nem figyeli az adott portot.



9.hét

Összefoglaló:

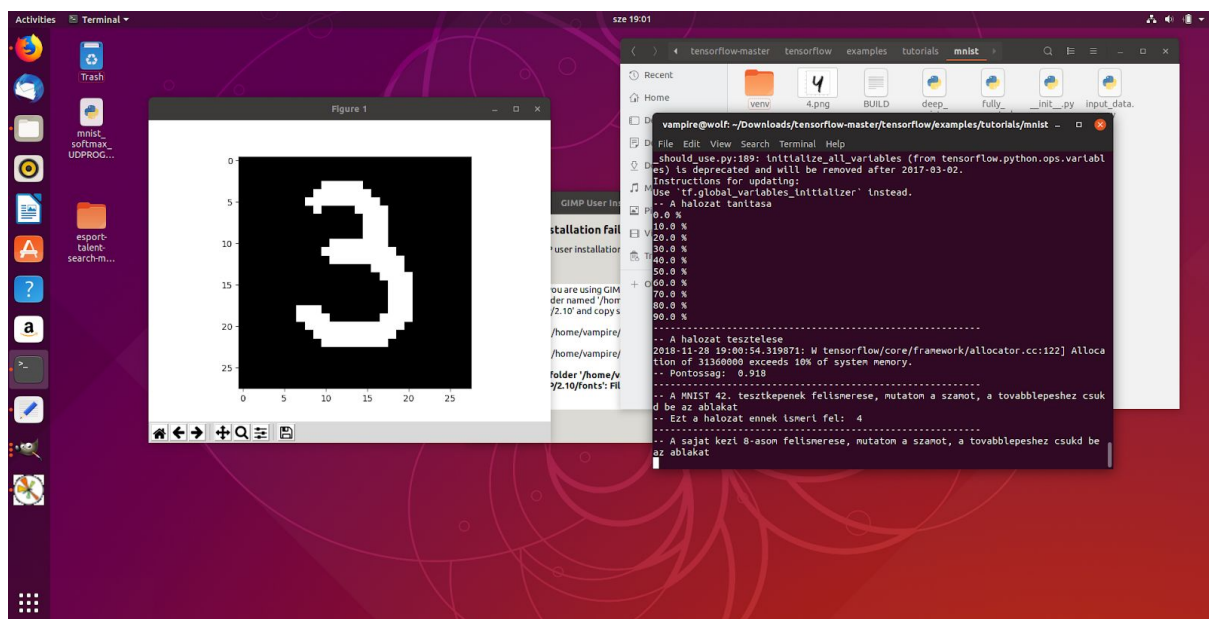
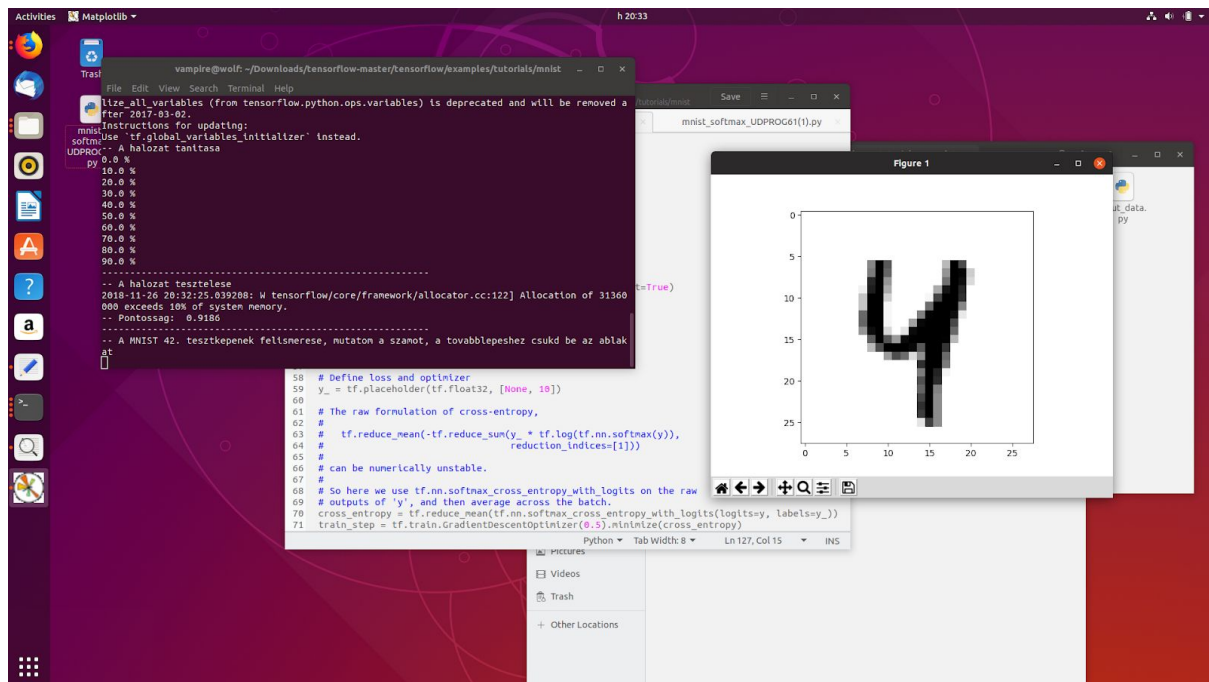
A TensorFlow rendszer kifejlesztésének előzményei a GoogleBrain gyökereiből származik. A GoogleBrain 2011-ben indult azzal az okkal, hogy az úgynevezett neurális hálók széles körben elterjedjenek. A korai demók eredményeképpen a DistBelief rendszer készült el széles körű kutatásokra. Maga a TensorFlow egy úgynevezett szoftverkönyvtár gépi tanulási algoritmusok végrehajtására és leírására. A neurális hálók kutatása több évtizedes múltra nyúlik vissza. Hirnévre pedig a beszédfeldolgozásnak és képfelismerésnek nyert nagyobb körű nyilvánosságot. Magát az algoritmust rengeteg dolog megvalósítására lehet alkalmazni, úgy mint a számítógépes támadások elleni harcban, a beszédfelismerésben, az információ kinyerésben mint a Google fordítója, ami képes egy arab szövegű leírást angol nyelvre fordítani, vagy a növekvő területű robotikában. A gyakorlatban rengeteg példát láthatunk még a Google térképben, a képfelismerőben stb. A Google kutatócsoport rengeteg más kutatócsoporttal dolgozik. A mérnökök rengeteg energiát fordítottak abban, hogy a TensorFlow algoritmus egyrészt

flexibilis legyen a kutatás számára, másrészt a valós alkalmazásokkal szemben támasztott követelményeknek megfelelő hatékony és robusztus legyen.

A TensorFlow kiszámítást egy irányított gráf írja le. Adatfolyam a gráf élével valósul meg. A TensorFlow gráfban mindegyik csúcs egy műveletet reprezentálhat és mindegyik csúcsnak lehet nulla vagy több inputja, ugyanúgy nulla vagy több outputja. A gráf normál élei mentén áramló értékek tenzorok, tetszőleges dimenziójú vektorok. Egy-egy elem fajtáját a gráf létrehozásával specifikálják. Lehetnek a gráfban speciális élek is, amelyek kontrol célokat szolgálnak, nem történik adatáramlás. A TensorFlow algoritmus tartalmaz még optimalizálást a számítási gráfban előforduló zajsűrés kiszűrésére, a kernel implementáció kiválasztására az adatbevitelre a memóriahasználatra.

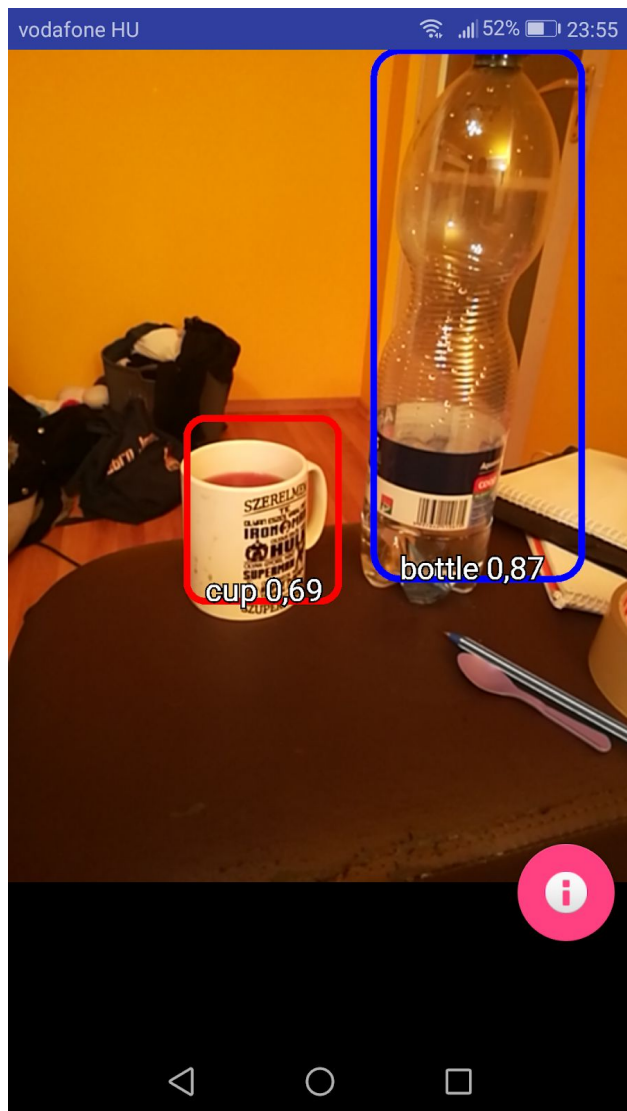
Mnlt:

A TensorFlow a Google tulajdonában lévő mély tanuló algoritmus bővebben az összefoglalóban fejtem ki. Az algoritmus egy 28x28 pixelből álló képet vár, ami fekete háttérű, és fehérrel van ráírva a mi általunk létrehozott 0-tól 9-ig terjedő számjegy. Ez lényegében egy képfelismerő algoritmus. Telepíteni kellett TensorFlow-t, majd ezt követően `mnist_softmax_UDPROG61.py` nevű file-t létre kellett hozni, majd amit a feladat leírásában megadott linken kódsorokat kellett másolni.



Andorid Telefonra TF:

Andorid Telefonra TF: egy hétköznapi képfelismerő algoritmus. Amely nem csak kiírja hogy milyen tárgyat ismer fel bekeretezve hanem annak valószínűségét is egy 0-1 terjedő skálán.



* A feladatok megoldásában: Varga Sándor Mátyás, Keserű Kristóf, Péter Erdős Udprog közösség, keresztapukám segített.