

Jegyzökönyv

Szilágyi-Czumbil Ede Balázs

Jegyzőkönyv

Szilágyi-Czumbil Ede Balázs

Table of Contents

I.	1
1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroway!”	5
OO szemlélet	5
„Gagyí”:	6
Yoda	8
4. 2. hét - „Helló, Liskov!”	11
Liskov helyettesítés sértése	11
Szülő-gyerek	12
Hello, Android!	15
5. 3. hét - „Helló, Mandelbrot!”	19
Reverse engineering UML osztálydiagram	19
Forward engineering UML osztálydiagram	20
BPMN	20
6. 4. hét - „Helló, Chomsky!”	22
Encoding	22
l33t	22
Fullscreen	22
7. 5. hét - „Helló, Stroustrup!”	23
Másoló-mozgató szemantika	23
Változó argumentumszámú ctor	26
Összefoglaló	26
8. 6. hét - „Helló, Gödel!”	29
STL map érték szerinti rendezése	29
Alternatív Tabella rendezése	29
9. 7. hét - „Helló, !”	30
FUTURE tevékenység editor	30
OOCWC Boost ASIO hálózatkezelése	30
BrainB	30
10. 8. hét - „Helló, Schwarzenegger!”	31
Port scan	31
Android Játék	31
11. 9. hét - „Helló, Calvin!”	32
MNIST	32
Android telefonra a TF objektum detektálója	32

List of Tables

7.1. Copy ctor vs Move ctor	28
-----------------------------------	----

Part I.

Table of Contents

1. Infók	3
2. 0. hét - „Helló, Berners-Lee!	4
Java, C++ összehasonlítás	4
Python	4
3. 1. hét - „Helló, Arroy!	5
OO szemlélet	5
„Gagy!	6
Yoda	8
4. 2. hét - „Helló, Liskov!	11
Liskov helyettesítés sértése	11
Szülő-gyerek	12
Hello, Android!	15
5. 3. hét - „Helló, Mandelbrot!	19
Reverse engineering UML osztálydiagram	19
Forward engineering UML osztálydiagram	20
BPMN	20
6. 4. hét - „Helló, Chomsky!	22
Encoding	22
l33t	22
Fullscreen	22
7. 5. hét - „Helló, Stroustrup!	23
Másoló-mozgató szemantika	23
Változó argumentumszámú ctor	26
Összefoglaló	26
8. 6. hét - „Helló, Gödel!	29
STL map érték szerinti rendezése	29
Alternatív Tabella rendezése	29
9. 7. hét - „Helló, !	30
FUTURE tevékenység editor	30
OOCWC Boost ASIO hálózatzelése	30
BrainB	30
10. 8. hét - „Helló, Schwarzenegger!	31
Port scan	31
Android Játék	31
11. 9. hét - „Helló, Calvin!	32
MNIST	32
Android telefonra a TF objektum detektálója	32

Chapter 1. Infók

Neptun kód: CMY9W3

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

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

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

Java, C++ összehasonlítás

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

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

Python

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

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

Chapter 3. 1. hét - „Helló, Arroway!”

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

OO szemlélet

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

A kód:

```
public class PolarGenerator {

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator() {
        nincsTarolt = true;
    }

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

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

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

            tarolt = r * v2;

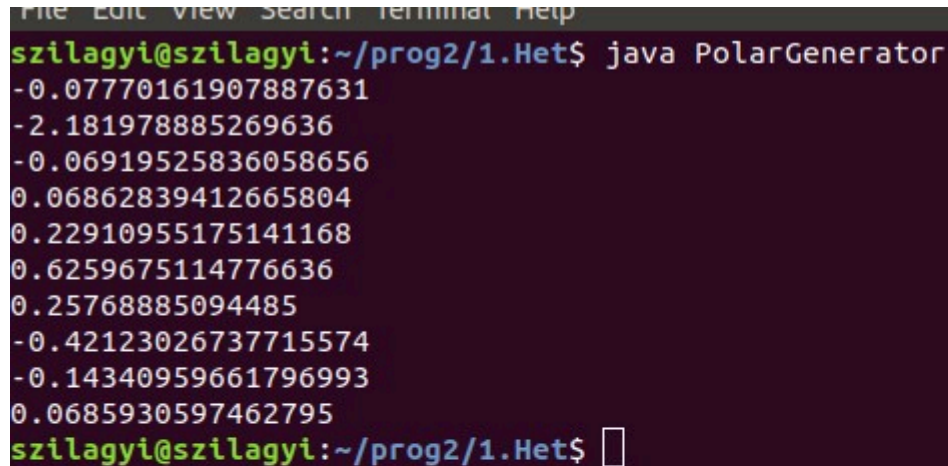
            nincsTarolt = !nincsTarolt;
            return r * v1;
        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

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

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

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

Miután futtatuk:



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

„Gagyi”:

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

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

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

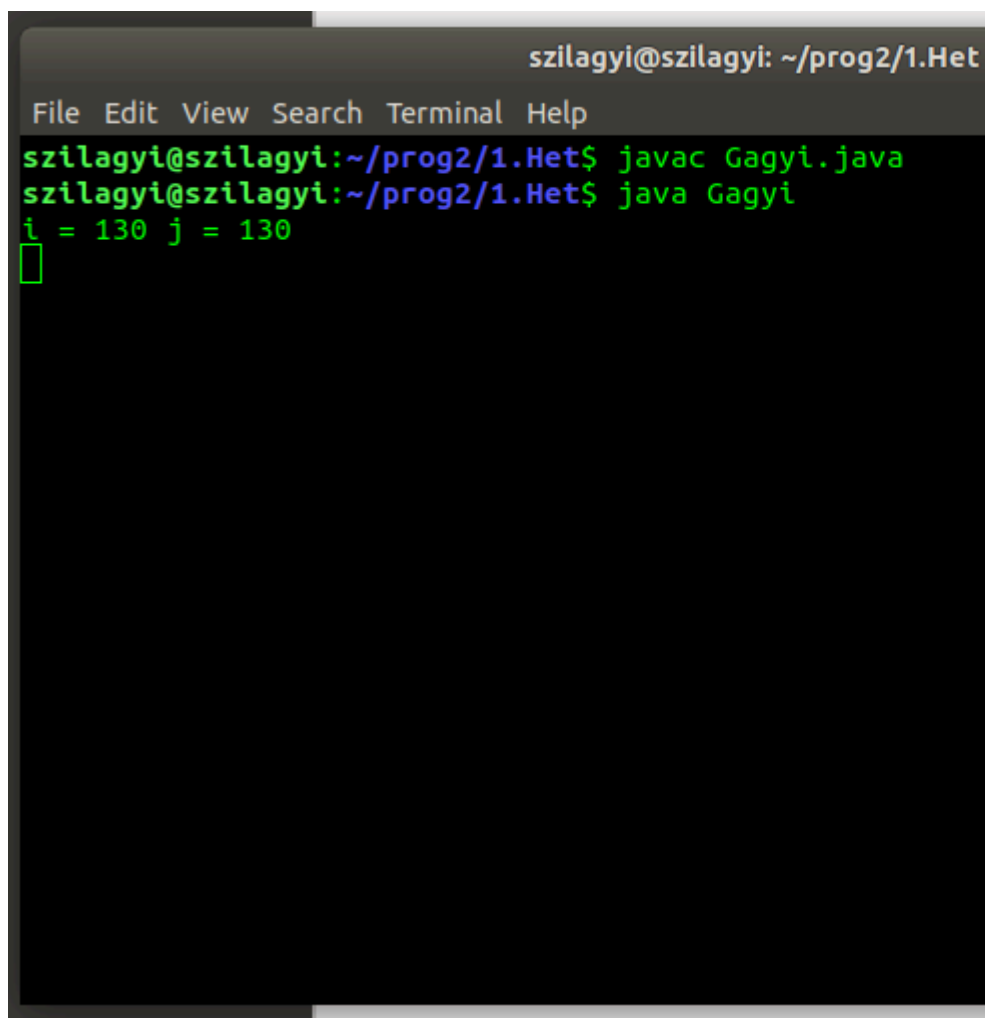
E miatt ha a programunkban a [-128,127] intervallumon kívüli értéket adunk meg akkor az egyenlőség hamis lesz mivel két új objektum fog létrejönni és emiatt végtelen ciklust kapunk.

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

a program ami végtelen ciklust ad:

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

lefuttatva:



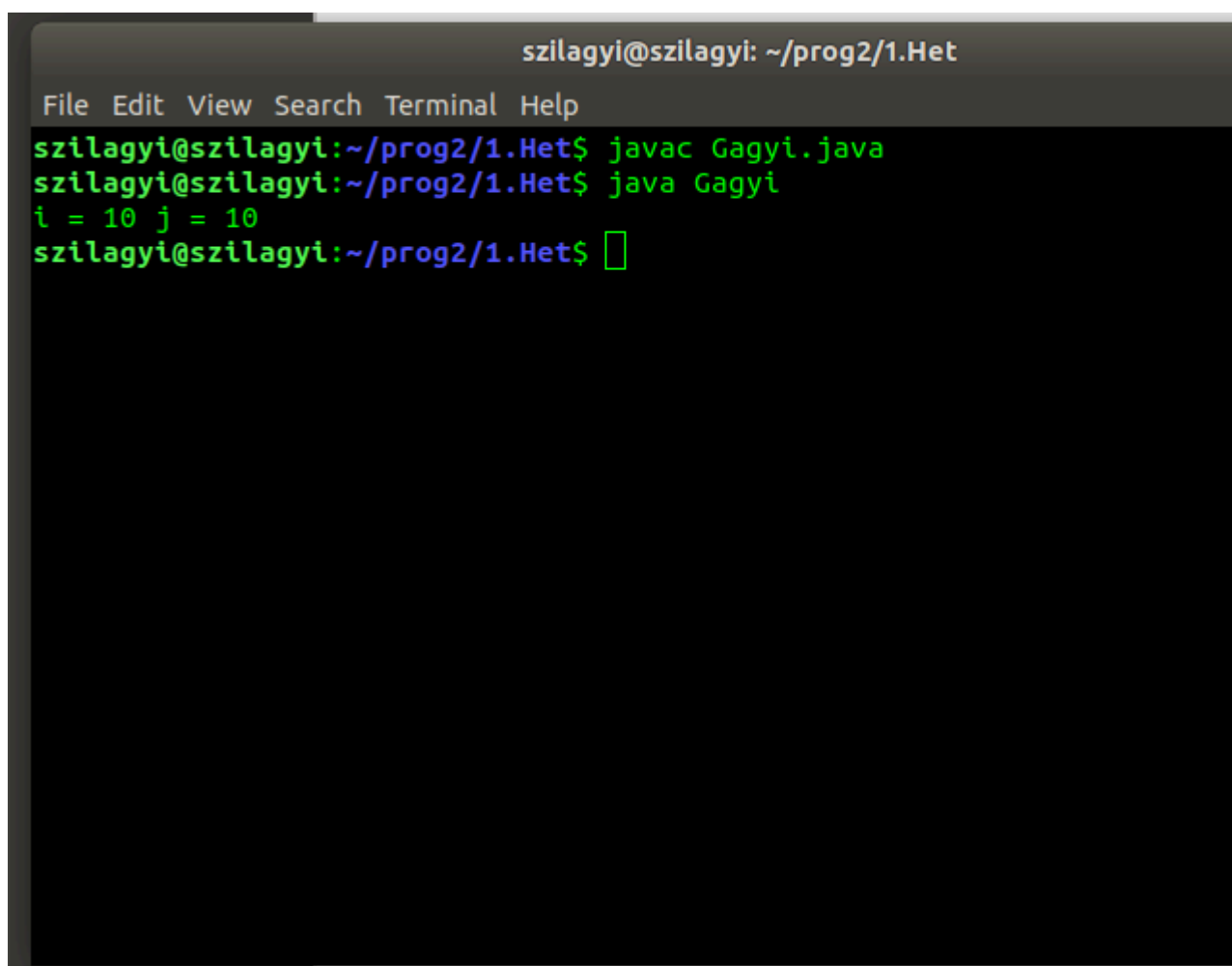
```
szilagyi@szilagyi: ~/prog2/1.Het  
File Edit View Search Terminal Help  
szilagyi@szilagyi:~/prog2/1.Het$ javac Gagyi.java  
szilagyi@szilagyi:~/prog2/1.Het$ java Gagyi  
i = 130 j = 130  
█
```

a program ami nem végtelen ciklust ad:

```
public class Gagyi {
```

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

lefuttatva:



```
szilagyi@szilagyi: ~/prog2/1.Het  
File Edit View Search Terminal Help  
szilagyi@szilagyi:~/prog2/1.Het$ javac Gagy1.java  
szilagyi@szilagyi:~/prog2/1.Het$ java Gagy1  
i = 10 j = 10  
szilagyi@szilagyi:~/prog2/1.Het$
```

Yoda

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

A kód:

```
public class Yoda {
```

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

    final String str = null;

    try {
        if(str.equals("...")) {
            //Do something
        }
        System.out.println("1. Success");

    } catch(Exception e) {
        System.err.println(e.getMessage());
    }

    try {
        if("..." equals(str)) {
            //Do something
        }
        System.out.println("2. Success");

    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

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

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

Lefuttatva:

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

Chapter 4. 2. hét - „Helló, Liskov!”

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

Liskov helyettesítés sértése

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

C++ kódcsipet amely megsérti a Liskov elvet:

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

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

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

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

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

    Sas sas;
    program.fgv ( sas );

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

Így lenne helyes:

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

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

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

class Sas : public RepuloMadar
{};

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

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

    Sas sas;
    program.fgv ( sas );

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

Szülo-gyerek

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

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

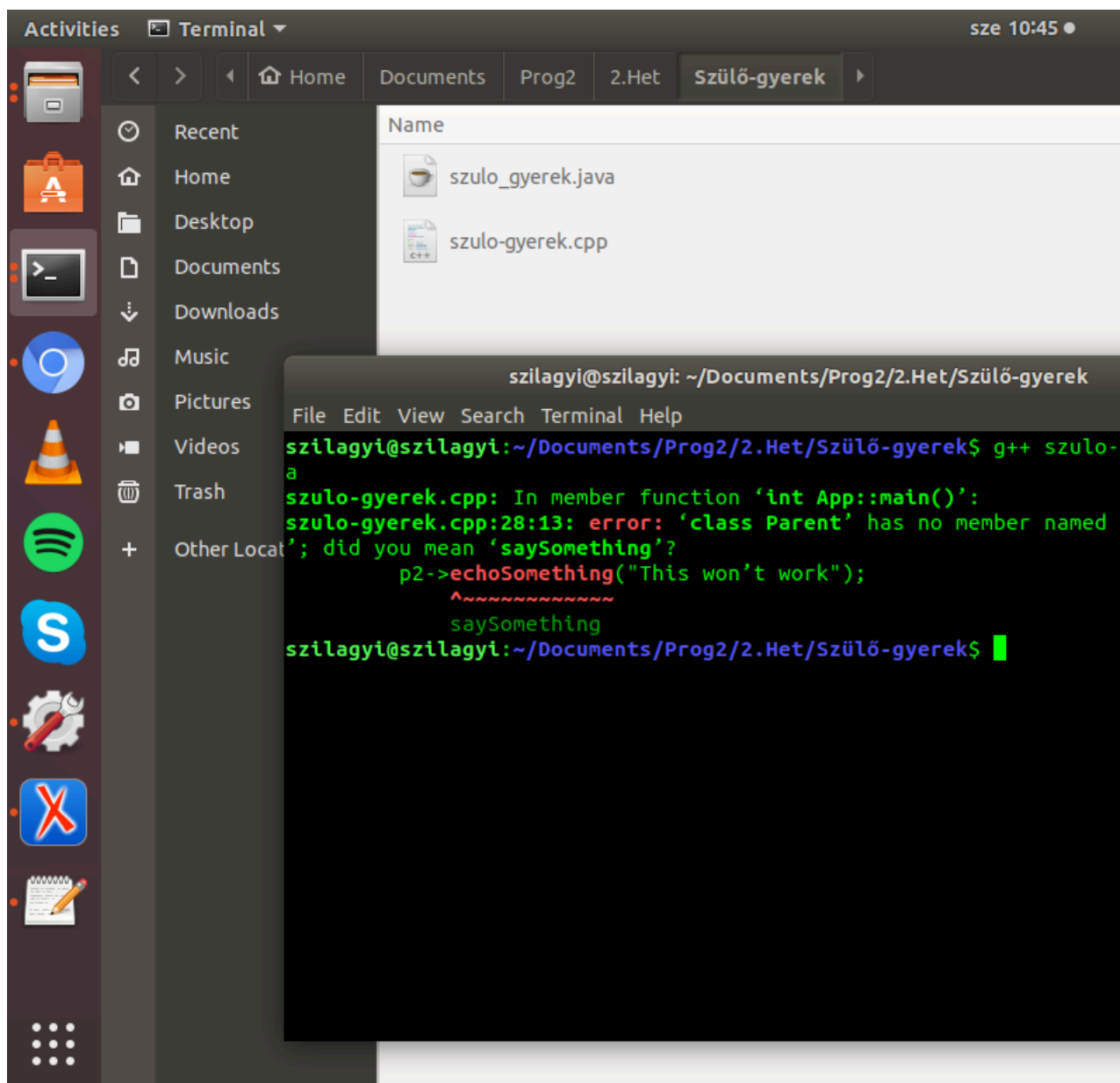
A nem Osök által definiált metódusokhoz nem férhetünk hozzá, hacsak nem # downcastoljuk az adott objektumot a tényleges típusára. Ez esetben viszont megsértjük az eloz# o feladatban ismertetett Liskov-elvet.

C++-ban:

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



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



Javában:

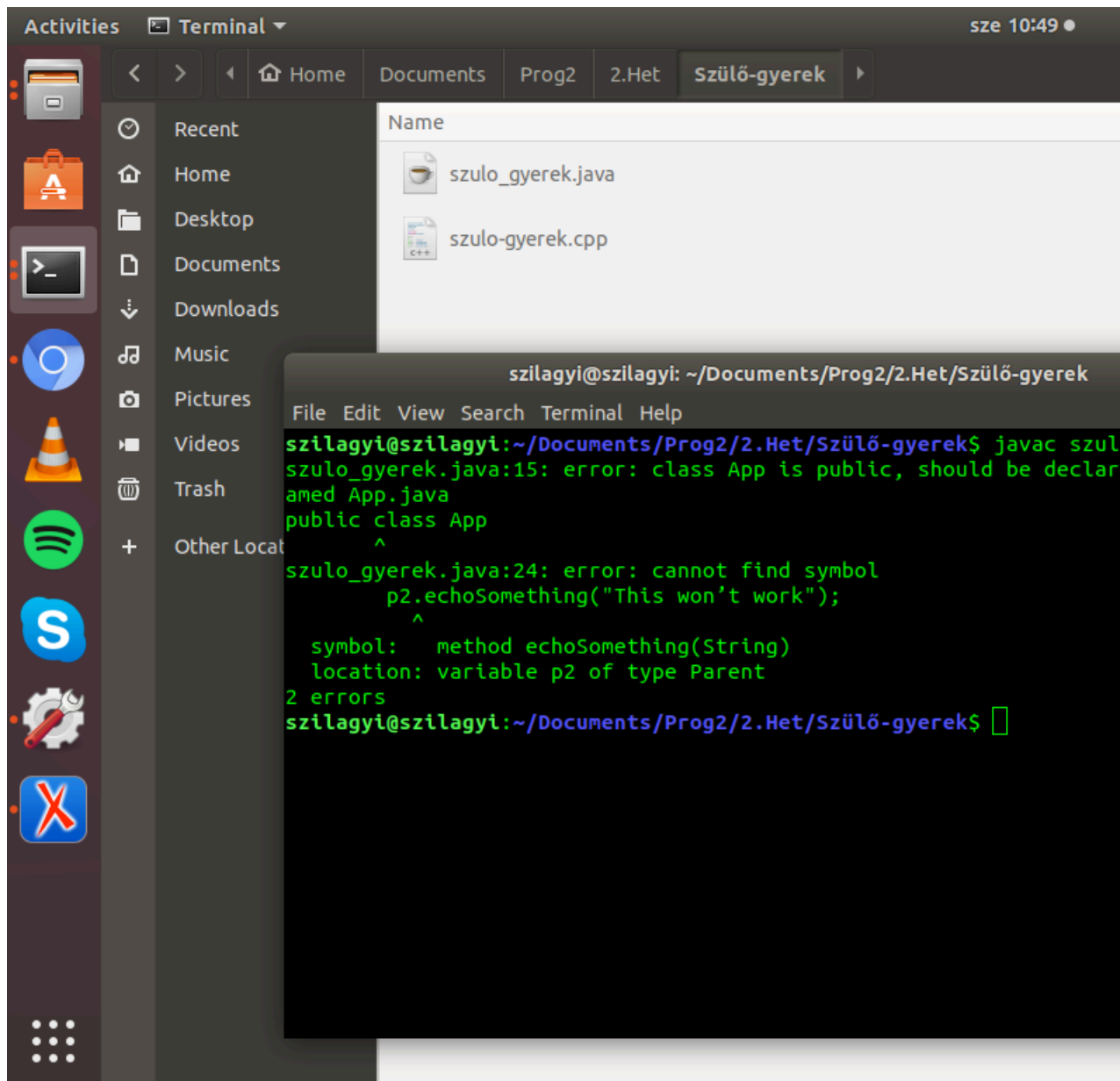
```
class Parent
{
    public void saySomething()
    {
        System.out.println("Parent says: BLA BLA BLA");
    }
}
class Child extends Parent
{
    public void echoSomething(String msg)
    {
        System.out.println(msg);
    }
}
public class App
{

```

```

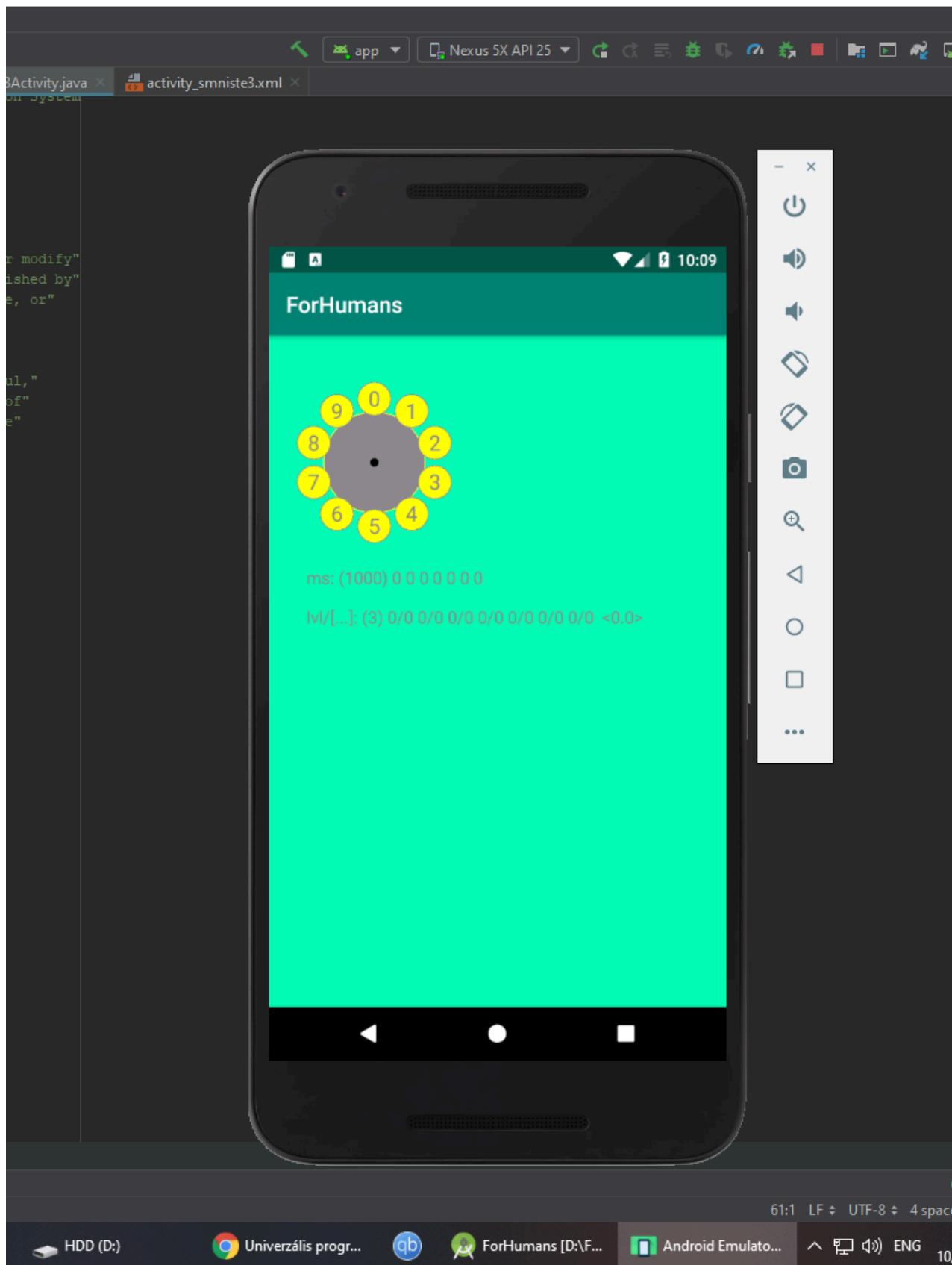
public static void main(String[] args)
{
    Parent p = new Parent();
    Parent p2 = new Child();
    System.out.println("Invoking method of parent");
    p.saySomething();
    System.out.println("Invoking method of child through parent ref");
    p2.echoSomething("This won't work");
}
}

```



Hello, Android!

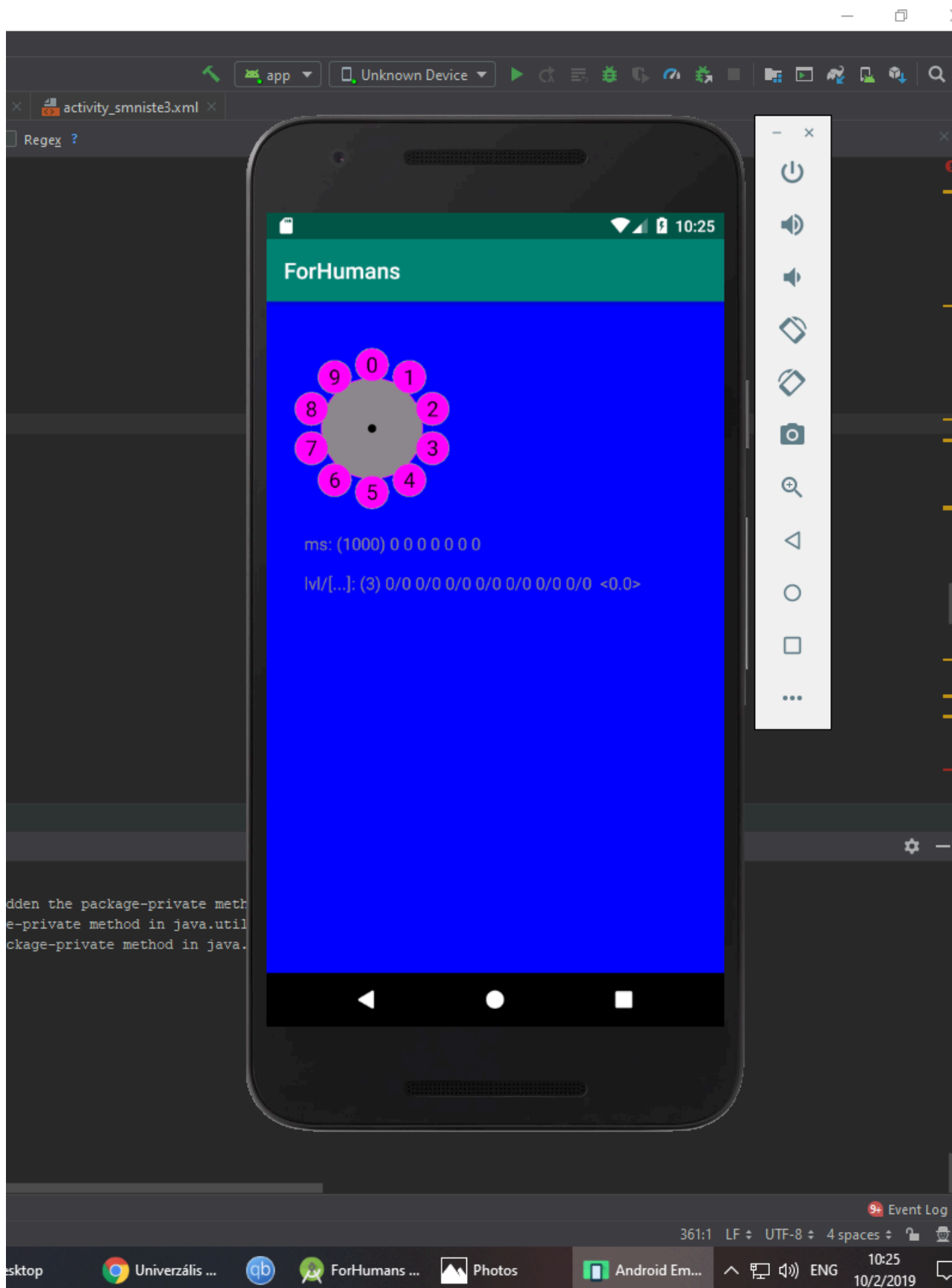
Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.



Márcsak a színvilágot kell megváltoztatni a feladat szerint: Ezt megtehetjük a SMNISTSurface-View.java állományban a megfelelo#

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

s a többi változók értékeinek megváltoztatásával.



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

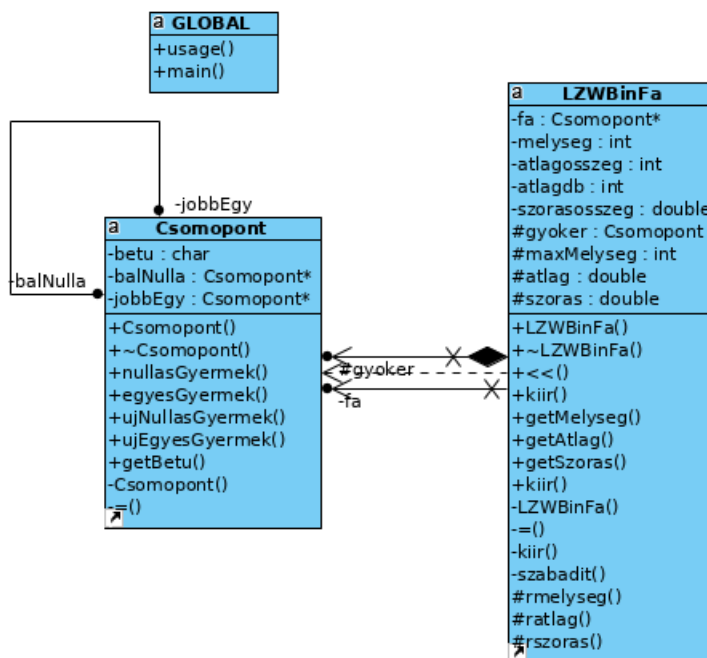
3. hét Modellez# eszközök és nyelvek. AZ UML és az UML osztálydiagramja.

Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatra a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs.

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

A megoldás forrása:



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

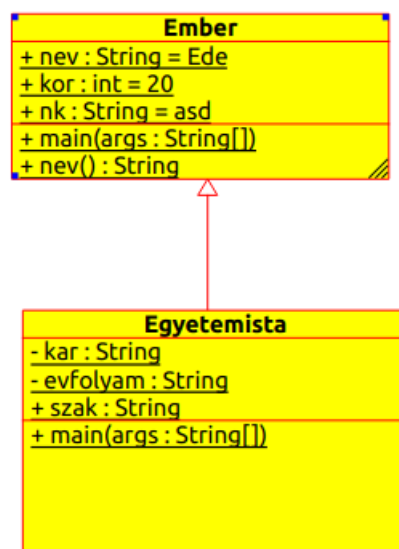
Ebben a feladatban az LZWBInfa kódból lett elkészítve a diagramm. A diagrammon látható hogy milyen osztályokból épül fel a program és azok között milyen kapcsolat van. Továbbá látható az is hogy az osztályok milyen változókkal és metódusokkal rendelkeznek. Mint látható egy osztály két részre van osztva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

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

Forward engineering UML osztálydiagram

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

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



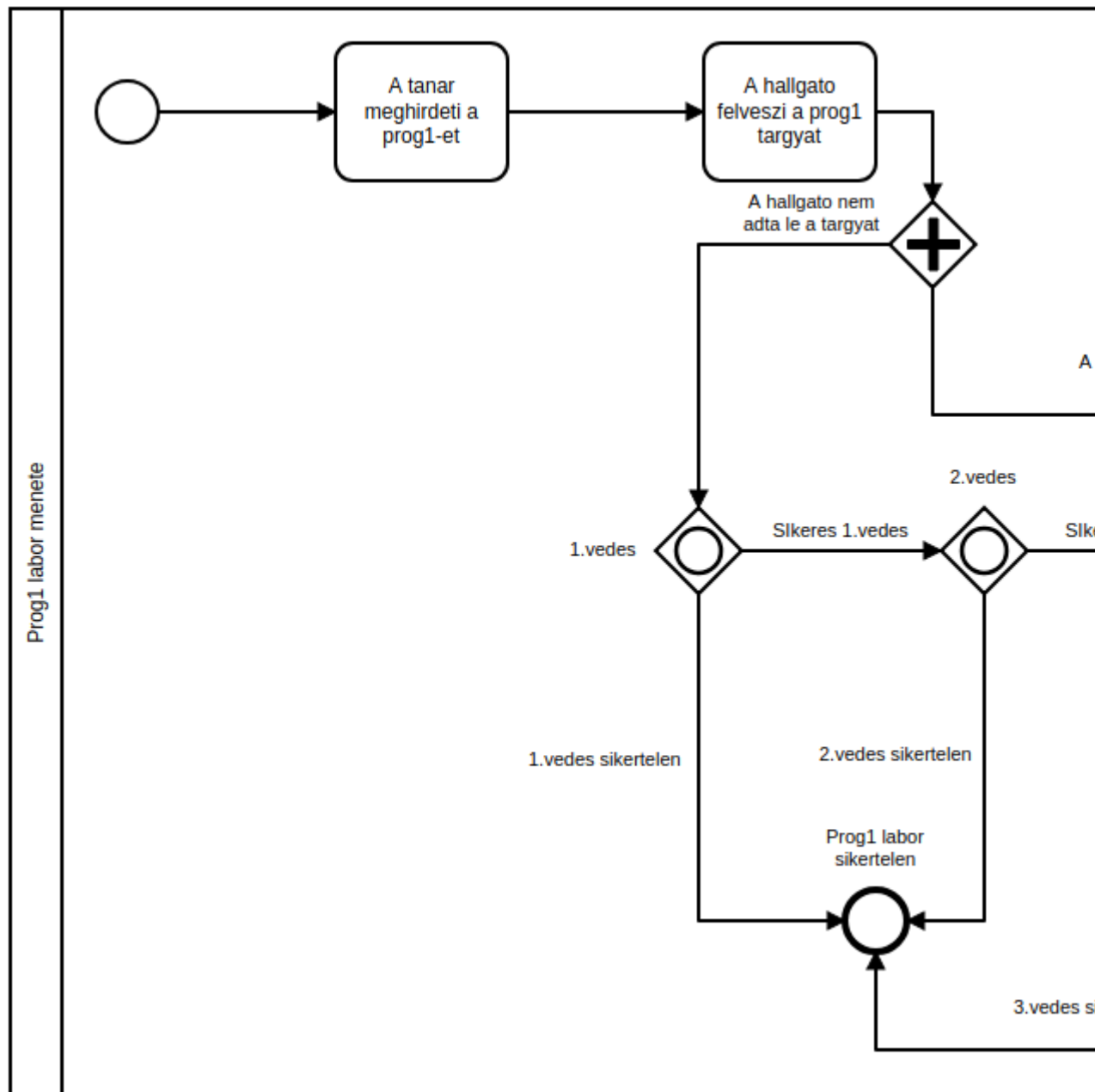
BPMN

Rajzoljunk le egy tevékenységet BPMN-ben!

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

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

Megoldás:



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

Chapter 6. 4. hét - „Helló, Chomsky!”

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

Encoding

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

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

I334d1c4

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

Fullscreen

Készítsünk egy teljes képernyős Java programot!

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

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

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

Másoló-mozgató szemantika

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

A kód:

```
#include <iostream>

class IntClass
{
public:
    int ertekek;

    IntClass () //parameter nélküli alap
    {
        ertekek = 0;
        std::cout << "Paraméter nélküli konstruktor.." << std::endl;
    }

    IntClass (int x)//parameteres
    {
        ertekek = x;
        std::cout << "Paraméteres konstruktor.." << std::endl;
    }

    void operator= (const IntClass &t) //egyenlőség
    {
        this->ertekek = t.ertekek;
        std::cout << "Egyenlőség operátor (osztály) túlterhelés.." << std::endl;
    }

    void operator= (int t)
    {
        this->ertekek = t;
        std::cout << "Egyenlőség operátor (int) túlterhelés.." << std::endl;
    }

    IntClass operator+ (const IntClass &t) //összeadás
    {
        IntClass x;
        x.ertekek = this->ertekek + t.ertekek;
        std::cout << "Összeadás operátor túlterhelés.." << std::endl;
        return x;
    }
}
```

```
IntClass operator- (const IntClass &t) //kivonás
{
    IntClass x;
    x.ertek = this->ertek - t.ertek;
    std::cout << "Kivonás operátor túlterhelés.." << std::endl;
    return x;
}

IntClass operator* (const IntClass &t) //szorzás
{
    IntClass x;
    x.ertek = this->ertek * t.ertek;
    std::cout << "Szorzás operátor túlterheles.." << std::endl;
    return x;
}

IntClass operator/ (const IntClass &t) //osztás
{
    IntClass x;
    x.ertek = this->ertek / t.ertek;
    std::cout << "Osztás operátor túlterheles.." << std::endl;
    return x;
}

void operator+= (const IntClass &t) //hozzáad
{
    this->ertek += t.ertek;
    std::cout << "+= operátor túlterheles.." << std::endl;
}

void operator-= (const IntClass &t) //kivon
{
    this->ertek -= t.ertek;
    std::cout << "-= operátor túlterheles.." << std::endl;
}

void operator++ (int) //növel
{
    ertek++;
    std::cout << "++ operátor túlterheles.." << std::endl;
}

void operator-- (int) //csökkent
{
    ertek--;
    std::cout << "-- operátor túlterheles.." << std::endl;
}

~IntClass()
{
    std::cout << "Destruktor.." << std::endl;
}

};

int
main()
{
```

```
IntClass x, y, z = 5;
x = 10;
std::cout << x.ertek << std::endl;

x = y;
std::cout << x.ertek << std::endl;

x = y + z;
std::cout << x.ertek << std::endl;

x = y - z;
std::cout << x.ertek << std::endl;

x = y * z;
std::cout << x.ertek << std::endl;

x = y / z;
std::cout << x.ertek << std::endl;

x += z;
std::cout << x.ertek << std::endl;

x -= z;
std::cout << x.ertek << std::endl;

x++;
std::cout << x.ertek << std::endl;

x--;
std::cout << x.ertek << std::endl;

return 0;
}
```

Lefuttatva:

```

File Edit View Search Terminal Help
szilagyi@szilagyi:~/prog2/5.het/Másoló-mozgató szemantika$ ./a.out
Paraméter nélküli konstruktor..
Paraméter nélküli konstruktor..
Paraméteres konstruktor..
Egyenlőség operátor (int) túlterhelés..
10
Egyenlőség operátor (osztály) túlterhelés..
0
Paraméter nélküli konstruktor..
Összeadás operátor túlterhelés..
Egyenlőség operátor (osztály) túlterhelés..
Destruktor..
5
Paraméter nélküli konstruktor..
Kivonás operátor túlterhelés..
Egyenlőség operátor (osztály) túlterhelés..
Destruktor..
-5
Paraméter nélküli konstruktor..
Szorzás operátor túlterheles..
Egyenlőség operátor (osztály) túlterhelés..
Destruktor..
0
Paraméter nélküli konstruktor..
Osztás operátor túlterheles..
Egyenlőség operátor (osztály) túlterhelés..
Destruktor..
0
+= operátor túlterheles..
5
-= operátor túlterheles..
0
++ operátor túlterheles..
1
-- operátor túlterheles..
0
Destruktor..
Destruktor..

```

Változó argumentumszámú ctor

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

Összefoglaló

Az el#z# 4 feladat egyikér#l írnj egy 1 oldalas bemutató „,”esszé szöveget!

Másoló-mozgató szemantika

1. Másoló szemantika:

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

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

- Destruktort,
- Másoló konstruktort,
- Vagy másoló értékadást

Akkor nagy valószínűséggel mindhármát implementálni kell.

Példa Rule of three-re:

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

2.Mozgató szemantika:

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

Példa:

```
#include<bits/stdc++.h>
int main()
{
    std::vector<int> vec1 {1, 2, 3, 4, 5};
    std::vector<int> vec2 {6, 6, 6, 6, 6};

    std::cout << "1.vektor elemei :";
    for(int i = 0; i < vec1.size(); i++)
        std::cout << " " << vec1[i];
    std::cout << "\n";

    std::cout << "2.vektor elemei :";
    for(unsigned int i = 0; i < vec2.size(); i++)
        std::cout << " " << vec2[i];
    std::cout << "\n\n";

    // elso 3 elemet az 1.vektorbol move-olja a 2.vektor 1.elemetol
    std::move(vec1.begin(), vec1.begin() + 3, vec2.begin());

    std::cout << "2.vektor elemei std::move utan:";
    for(unsigned int i = 0; i < vec2.size(); i++)
        std::cout << " " << vec2[i];
    std::cout << "\n";

    return 0;
}
```

Lefuttatva:

```

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

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

```

Table 7.1. Copy ctor vs Move ctor

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

Chapter 8. 6. hét - „Helló, Gödel!”

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

STL map érték szerinti rendezése

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

Alternatív Tabella rendezése

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

Chapter 9. 7. hét - „Helló, !”

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

FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon!

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

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

OOCWC Boost ASIO hálózatkezelése

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

BrainB

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

Chapter 10. 8. hét - „Helló, Schwarzenegger!”

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

Port scan

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

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

Android Játék

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

Chapter 11. 9. hét - „Helló, Calvin!”

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

MNIST

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

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

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

Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!