

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

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

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Szűcs, Levente	2019. október 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

Tartalomjegyzék

I. Második felvonás	1
1. Helló, Berners-Lee!	3
1.1. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I--II.II.	3
1.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. I--II.II.	3
2. Helló, Arroway!	5
2.1. Olvasónapló	5
2.2. OO szemlélet	5
2.3. Gagy	11
2.4. Yoda	13
3. Helló, Liskov!	15
3.1. Liskov helyettesítés sértése	15
3.2. Szülő-gyerek	16
3.3. Anti OO	18
3.4. Ciklomatikus komplexitás	19
4. Helló, Mandelbrot!	20
4.1. Reverse engineering UML osztálydiagram	20
4.2. Forward engineering UML osztálydiagram	21
4.3. BPMN	22
4.4. BPEL Helló, Világ! egy visszhang folyamat	23

II. Irodalomjegyzék	25
4.5. Általános	26
4.6. C	26
4.7. C++	26
4.8. Lisp	26

DRAFT

Táblázatok jegyzéke

3.1. Összehasonlítás	19
--------------------------------	----

Előszó

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

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

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

Hogyan forgasd

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

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Második felvonás

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

DRAFT

1. fejezet

Helló, Berners-Lee!

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

A Java és a C++ magasszintű, objektumorientált programozási nyelv. Nagyon sok hasonlóság van a két nyelvben, hiszen a Java nagyon sok dolgot vett át a C++ tól. A Java viszont újabb ezért a fejlesztők sokat dolgoztak azon, hogy a Java kódok megbízhatóbbak, biztonságosabb és emellett platformfüggetlenek legyenek. Mint említettem, a két nyelv objektumorientált, de a C++ nem kötelez arra hogy osztályokba dolgozzunk, ezzel szembe a Javában csak objektumorientáltan lehet dolgozni.

A Java és a C++ szintaxisa nagyon hasonló, hiszen a Java a C és a C++ szintaxisán alapszik. Természetesen vannak apróbb különbségek. Később kifejtem...

A két nyelv mint említettem nagyon hasonlít azonban van egy nagy különbség. A memóriakezelés a C++-ban manuális míg Javában a memóriakezelést a Garbage collection "szemét gyűjtő" végzi ezt a munkát. A Javában a rendszer önmaga kezeli a memóriát és ha rendszer úgy érzi, hogy betelik a memória, akkor felszabadít memóriát. A C++-ban a programozónak kell a memóriát kezelnie. A C++-ban destruktorkat alkalmazunk ahhoz, hogy memóriát szabadítsunk fel. Természetesen mindkettőnek megvannak előnyei, hátrányai. Javában az előny az, hogy nem kell nekünk kezelni ezeket a memória felszabadításokat, amely egy nagyobb projektnél egy nagyon kényelmes funkció, azonban mi ezt nem tudjuk irányítani, ezért lehetséges az, hogy feleslegesen tud nagyobb méretű memóriát foglalni a rendszer egy kisebb programhoz, mint amennyire aktuálisan szükségünk van rá.

1.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. I--II.II.

Guido van Rossum 1990-ben a Pythont, amely egy általános célú magasszintű programozási nyelv. Ez a nyelv a fejlesztők számára rengeteg pozitív tulajdonsággal rendelkezik ezek közül pl:dinamikus,objektumorientált és a platformfüggetlenség. Python tulajdonképpen egy szkriptnyelv, azonban rendkívül sok csomagot és beépített eljárását tartalmaz, ezért összetett alkalmazások készítésére is alkalmas.

Nagyon sok platformokon érhető el pl: Win, Mac, Unix, S60, UIQ, Iphone. A Python egy köztes nyelv, nincs szükség fordításra és linkelésre. Az értelmező (interpreter) interaktívan is használható amelynek segítségével például egyszerűen kezelhetünk (throw-away) programokat is bizonyos funkciók kipróbálására.

A nyelv legfőbb jellemzője, hogy behúzásalapú a szintaxisa. A programban szereplő állításokat az azonos szintű behúzásokkal tudjuk csoportokba szervezni, nincs szükség kapcsos zárójelre vagy explicit kulcsszavakra(pl.begin,end). Fontos, hogy a behúzásokat egységesen kezeljük, tehát vagy mindenhol tabot, vagy egységesen szóközt használjuk.

A nyelv további sajátossága, hogy a sor végéig tart egy utasítás, nincs szükség a megszokott ';' használatára. Ha egy utasítás több sorban fér csak el, akkor ezt a sor végére '\n' (backslash) jellel lehet jelezni. Amennyiben nem zártunk be minden nyitott zárójelet akkor az utasítás folytatósorának veszi a következő sort. Az értelmező minden egyes sort úgynevezett tokenekre bont, amelyek közt tetszőleges üres whitespace karakter lehet. Tokenek fajtái:azonosító kulcsszó,operátor,literál. Kulcsszavakat itt is is alkalmazunk pl : lambda,return,while,try stb.

Pythonban minden adatot objektumok reprezentálnak. Pythonban nincs szükség a változók típusainak explicit megadására,a rendszer futási időben,automatikusan „kitalálja” a változók típusát. Az adattípusok a következők lehetnek: számok, sztringek,ennek (tuples,n-es), listák, szótárak. A számok lehetnek egészek, lebegőpontosak és komplex számok. Az egész számok lehetnek decimálisak,oktálisak.

2. fejezet

Helló, Arroway!

2.1. Olvasónapló

Az objektumorientált programozás lényegét az adatabsztrakció, öröklődés és a polimorfizmus szavakkal foglалhatjuk össze. Az öröklődés legegyszerűbb esete amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk.

A kiterjesztés jelentheti új műveletek és/vagy új változók bevezetését.

Az osztályok rokonsági viszonyainak összességét osztályhierarchiának hívjuk amit gyakran mint fentről lefelé nöő fát ábrázolnak az Object osztályból ágaztatva.

Az Object osztály egy kiemelt osztály amely java.langba csomagba tartozik. Object implicit módon minden osztálynak szülője amely nem más osztályból származik vagyis amelynek definíciójában nem adjuk meg az extends tagot. Object minden más osztálynak az őse. Az object típusú változók felelnek meg a Java nyelvben a típus nélküli mutatóknak, hiszen tetszőleges típusú objektumra (beleértve tömböket is) hivatkozhatnak.

Minden elemre külön lehet szabályozni annak láthatóságát. Public a külvilág számára látható. Egy elem lehet csak a leszármazottak(protected) vagy senki más számára sem látható(private) is. Ha a megjelölés elmarad, akkor az adott elem csak a forrásszöveg a környezetében, az adott csomagban lesz látható, a külvilág számára nem.

A nyelvnek van egy másik fontos eleme, amely nem része az osztályoznak, ez az interfész. Az interfész már az Objective-C nyelvben is szerepelt protokoll néven, ezt a Java fejlesztői is átvették és kissé módosították. Egy interfész egy új referencia típus, absztrakt metódusok deklarációjának és konstans értékeknek az összessége. Az interfészek az egyik alapvető tulajdonsága, hogy benne található metódusok csak deklarálódnak, vagyis megvalósítás nélkül szerepelnek.

Az interfészek felületet definiálnak. Egy interfész tényleges használata az implemetációján keresztül történik. Egy osztály implementál egy interfészt, ha az összes interfész által specifikált metódushoz implemetációt ad. Ezáltal az absztrakt program konkréttá válik.

2.2. OO szemlélet

A legelső feladatunk amellyel foglalkoznunk kellett az a Polárgenerátor. Ezzel a Polárgenerátorral random számokat tudunk képezni. Először Javában majd C++ nyelven kellett megnézni ezeket a feladatokat a fólia jegyzeteinek köszönhetően, nem volt nehéz dolgunk.

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 g = new PolarGenerator();

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

Vegyük is sorjába a dolgokat.

```
public class PolarGenerator{
```

```
boolean nincsTarolt = true;  
double tarolt;
```

Ebbe az osztályunkba 2 változót fog tartalmazni. Amint láthatjuk, hogy a 2 változó típusa különböző, először is a nincsTarolt egy boolean típusú változó amelynek értéke vagy igaz vagy hamis lehet. Ez a változóba mondja meg, hogy van-e tárolt változónk vagy nincsen. A másik változóba a taroltba pedig a változó értékét adjuk meg tizedes törtben.

```
        public PolarGenerator() {  
            nincsTarolt = true;  
        }
```

Ezután ebbe konstruktorba a nincsTarolt-at igazgá tesszük.

```
        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;  
            }  
        }
```

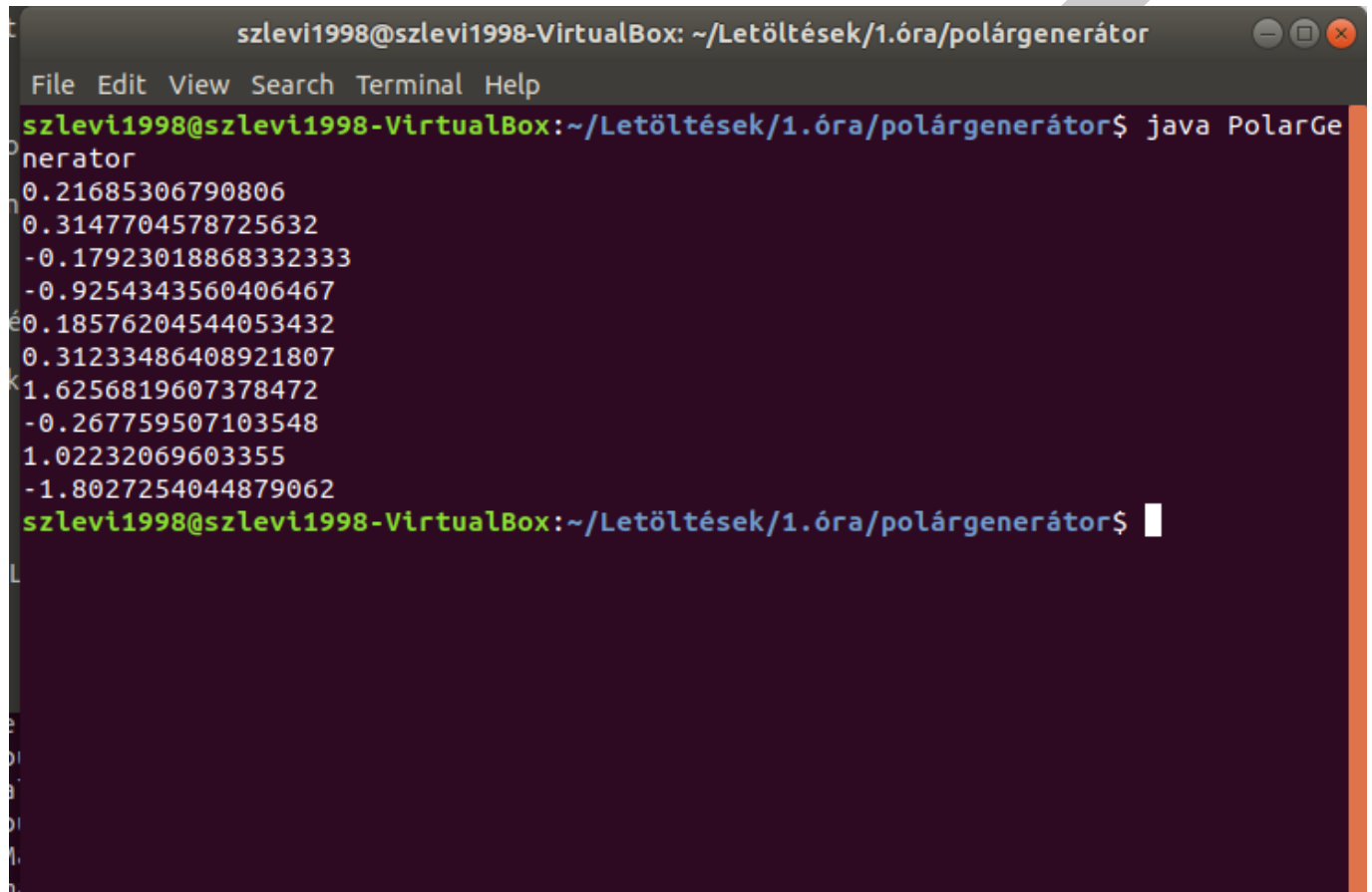
Ebbe a kovetkezo nevű függvény fut le a számítás. Ha nincs tárolt értékünk akkor 2 értéket számolunk. Akkor ha van tárolt értékünk, akkor pedig azt a tárolt értéket visszaadjuk.

```
        public static void main(String[] args) {  
  
            PolarGenerator g = new PolarGenerator();  
  
            for(int i = 0; i < 10; ++i)
```



```
System.out.println(g.kovetkezo());  
}
```

Ezután a mainben lefut a programunk. Példányosítjuk a PolárGenerátor osztályt. Ezután egy for ciklusban 10-szer hívjuk meg a kovetkezo() metódust. A program 10 darab random számot fog nekünk kiírni amit itt is láthatunk.



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/polárgenerátor  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ java PolarGe  
nerator  
0.21685306790806  
0.3147704578725632  
-0.17923018868332333  
-0.9254343560406467  
0.18576204544053432  
0.31233486408921807  
1.6256819607378472  
-0.267759507103548  
1.02232069603355  
-1.8027254044879062  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$
```

Ez a program nagyon hasonlít az eredeti, a JDK-s src fájljaiban is megtalálható random generátorra. Itt láthatjuk a Java.util.Random osztályban lévő Randomot és látható benne a hasonlóság.

```
public synchronized double nextGaussian()
{
    if (haveNextNextGaussian)
    {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    }
    double v1, v2, s;
    do
    {
        v1 = 2 * nextDouble() - 1; // Between -1.0 and 1.0.
        v2 = 2 * nextDouble() - 1; // Between -1.0 and 1.0.
        s = v1 * v1 + v2 * v2;
    }
    while (s >= 1);
    double norm = Math.sqrt(-2 * Math.log(s) / s);
    nextNextGaussian = v2 * norm;
    haveNextNextGaussian = true;
    return v1 * norm;
}
```

Ezután a programunkat megírtuk C++ -ban is. A kódok itt vannak:

Először is a header file:

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
#include <ctime>

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

private:
    bool nincsTarolt;
    double tarolt;
```

```
};  
#endif
```

Itt található a fájlunknak a header fájlja. Ebbe találhatóak meg az osztályok. Két elkülöníthető osztályunk van a public és a private. A public az osztályon kívül is elérhető. Ezzel szembe a private csak az adott osztályon belül érhető el.

A privateba megtalálható ugyan az a két változónk itt is az egyik változó bool a másik pedig szintén egy tört változó csak úgy mint a Javas kódunkban.

```
#include "polargen.h"  
  
double  
PolarGen::kovetkezo ()  
{  
    if (nincsTarolt)  
    {  
        double u1, u2, v1, v2, w;  
        do  
        {  
            u1 = std::rand () / (RAND_MAX + 1.0);  
            u2 = std::rand () / (RAND_MAX + 1.0);  
            v1 = 2 * u1 - 1;  
            v2 = 2 * u2 - 1;  
            w = v1* v1 + v2 * v2;  
        }  
        while(w > 1);  
  
        double r = std::sqrt ((-2 * std::log (w)) / w);  
  
        tarolt = r * v2;  
        nincsTarolt = !nincsTarolt;  
  
        return r * v1;  
    }  
    else  
    {  
        nincsTarolt = !nincsTarolt;  
        return tarolt;  
    }  
}
```

Amint látható a két nyelv között nincsen óriási különbség. Szintén a számítás csak úgy mint a Java-ban a kovetkezo függvényben megy létre. Abban az esetben, ha a nincsTarolt igaz akkor egy random törtszámot generál, ha hamis akkor pedig visszaadja az előzőleg tárolt értéket.

```
int main (int argc, char **argv)  
{  
    PolarGen pg;
```

```
for (int i = 0; i < 10; ++i)
    std::cout << pg.kovetkezo() << std::endl,

return 0;
}
```

Csak úgy mint a Javában, itt is létrehozuk a mainben a pg PolarGen osztályt és ezután, a for ciklus 10-szer lefut, és a kovetkezo() kiíratja a random számokat. Ezeket az értékeket láthatjuk:



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/polárgenerátor
File Edit View Search Terminal Help
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ g++ polargenteszt.cpp -o Polar
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$ ./Polar
1.67712
-0.734798
-1.47623
1.02756
-1.03194
-1.07785
-0.256456
0.710424
-0.339193
-0.744762
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/polárgenerátor$
```

2.3. Gagyí

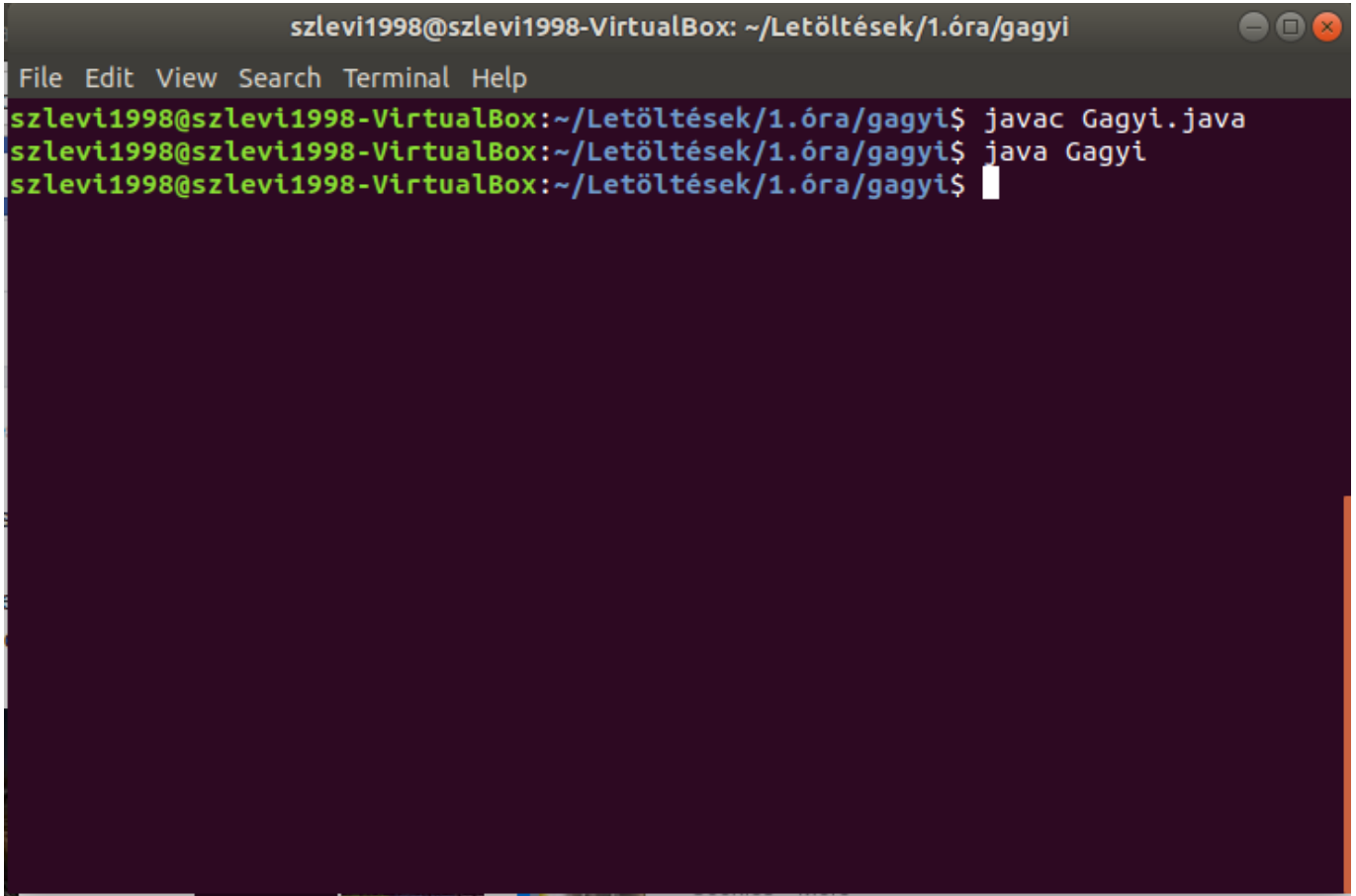
```
while(x <=t && x>=t && tt !=x);
```

Ebben a feladatban ezzel a kódcsipettel foglalkoztunk, a feladatunk az volt, hogy ezzel a példaprogrammal egyszer végtelen ciklust, más értékkel pedig ne okozzon.

```
Public class Gagyí
{
    public static void main(String[] args){
        Integer t = -128;
        Integer x = -128;

        while (x <=t && x>=t && tt !=x){
            System.out.println("yes");
        }
    }
}
```

Ebben a kódcsipetben két Integer objektumot hozunk létre. Fontos tudni, hogy az Integer objektumok egy előre készített poolból veszi az értékeit -128-tól 127-ig. Ilyenkor két objektumunk a memóriacímük megegyezik, azért mert a poolban lévő értékeknek a memóriacímük megegyezik. Ez a while ciklus hamis lesz, azért mert a != operátor feltétele teljesül, ezért nincs végtelen ciklus.



```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/gagyí
File Edit View Search Terminal Help
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$ javac Gagyi.java
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$ java Gagyi
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/gagyí$
```

Viszont abban az esetben, hogy ha az értékünk -129, akkor ez már nem fog teljesülni hiszen a -129 nem a pool része ezért a memóriacímük nem fog megegyezni, ezért a while ciklus igaz és ez végtelen ciklust hoz létre.

[illegible]

2.4. Yoda

Ebben a feladatban a Yoda feltételekkel foglalkozunk. Köztudott, hogy Yoda a Star Wars filmekben nyelvtanunkhoz képest fordítottan beszél, hiszen először az igéket használja, majd a tárgyat mondja. Pl: "I have a phone." Yoda "nyelvtannal" így hangzik : "Phone, I have."

Yoda feltételekben szokástól eltérően a változó értékét bal oldalra írjuk míg jobbra kerül maga a változó neve. Vannak olyan esetek amelyben hasznos tud lenni, azonban nem szokták legtöbbször alkalmazni. Hasznos tud lenni ha null pointer Exception hibákat akarunk elkerülni, azonban a veszélye így is fennáll, hogy ez a hiba későbbiekben fennáll. Ami előny, hogy akkor amikor a konstans bal oldalra kerül, akkor alapvetően nem változtatja meg a program működését. Hátrányok közé sorolható, mint említettem, hogy a null pointer Exception hibákat csak elrejt, hiba továbbra is fennállhat, illetve olvashatóság szempontjából sem a legideálisabb.

Megszokott szintaxis:

```
int yoda = 5;
if (yoda == 5){
System.out.println("this statement is true");
}
```

Yodás szintaxis

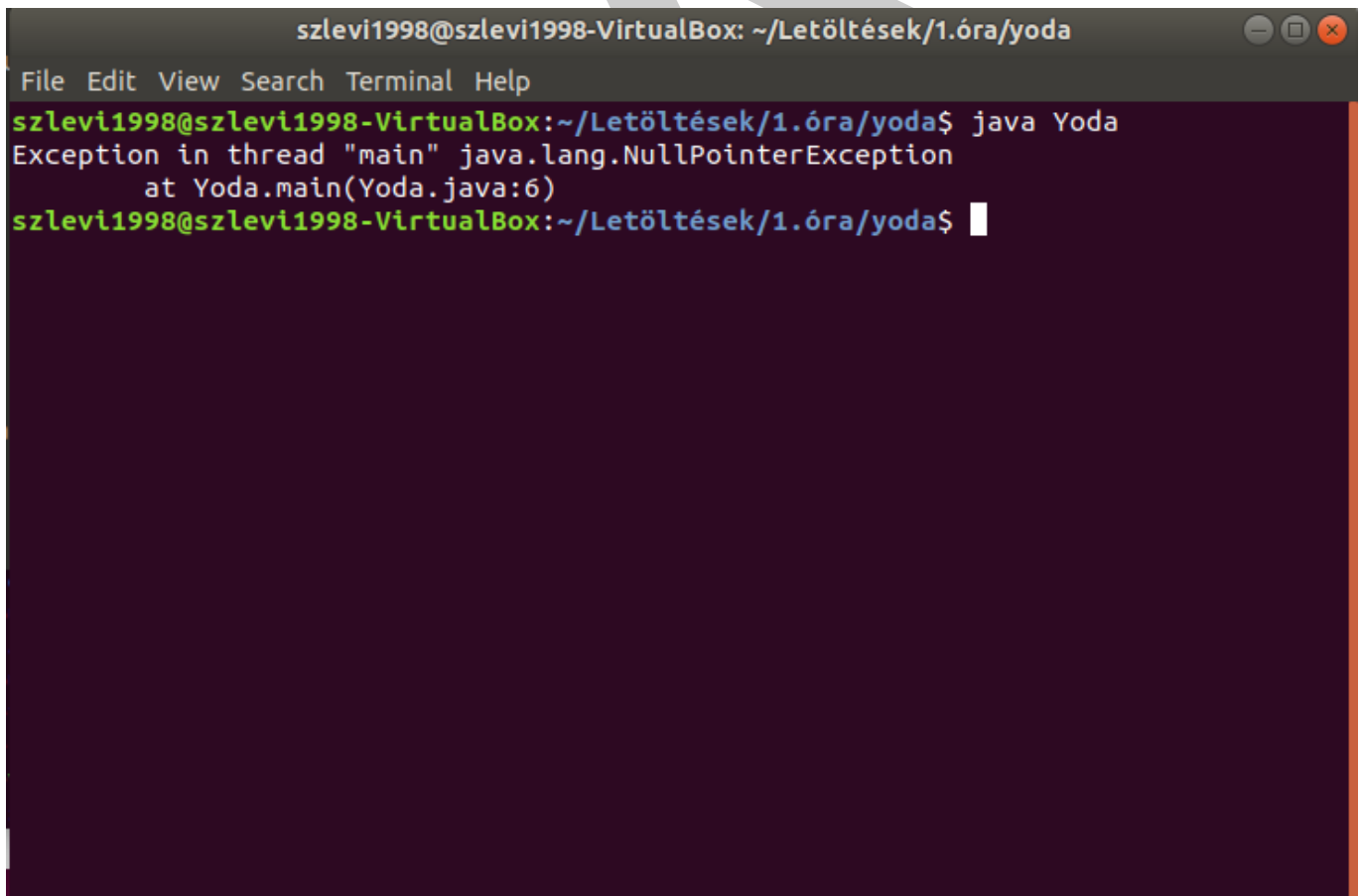
```
int yoda = 6;  
if (6 == yoda) {  
    System.out.println("This statement is also true");  
}
```

Amint láthatjuk hogy az érték és a változó megcserélődött ebbe a szintaxisban.

És akkor a kód:

```
public class Yoda  
{  
    public static void main(String[] args)  
    {  
        String Yoda = null;  
        if(Yoda.equals("something")) {  
            System.out.println("valami más");  
        }  
    }  
}
```

A kód nem követi a Yoda conditions és ezért a programunk leáll java.lang.nullPointerExceptionnel.



The screenshot shows a terminal window titled "szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/yoda". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The command prompt shows the user running "java Yoda". The output is an exception message: "Exception in thread "main" java.lang.NullPointerException at Yoda.main(Yoda.java:6)". The prompt then returns to the shell.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/1.óra/yoda  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/yoda$ java Yoda  
Exception in thread "main" java.lang.NullPointerException  
    at Yoda.main(Yoda.java:6)  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/1.óra/yoda$
```

3. fejezet

Helló, Liskov!

3.1. Liskov helyettesítés sértése

Ebben a feladatban a Liskov helyettesítés sértésére kellett programot írunk. A híres S.O.L.I.D programozásnak (Tiszta kódolás) az L betűje a Liskov-elv amely arról szól, hogy minden osztályt lehet helyettesíteni a leszármazott osztályával, amellyel a program alapvető működése nem változik. Ebben a feladatban segítségül kaptuk az Udprog repójában lévő forrást. Ez alapján csináltam egy saját programot. Íme :

```
class Jarmuvek{
public:
    virtual void gurul(){};
};

class Program {
public:

    void fgv ( Jarmuvek &jarmuvek ) {
        jarmuvek.gurul();
    }
};

class Car : public Jarmuvek
{};

class Hajo : public Jarmuvek
{};

int main ( int argc, char **argv )
{
    Program program;
    Jarmuvek jarmuvek;
    program.fgv ( jarmuvek);

    Car car;
    program.fgv ( car );
}
```



```
Hajo hajo;  
program.fgv ( hajo );  
  
}
```

Ebben a kódban a Járművek osztály az űsosztály és ebben a gyermekosztályok az autók és a hajók. Itt van egy gurul függvény, amely a problémát okozza, itt minden egyes gyerekosztályra utal azaz a Carra és a hajóra. A Carral semmi gond azonban, a hajó nem tud gurulni, viszont ez így nem igaz ezért a Liskov-elv sérül.

3.2. Szülő-gyerek

Ebben a feladatban a szülő és gyerek osztály kapcsolatáról foglalkozunk. Ez a feladat a polimorfizmussal és leginkább az öröklődéssel foglalkozik.

Az objektumorientált programozás lényegét az adatabsztrakció, öröklődés és a polimorfizmus szavakkal foglалhatjuk össze. Az öröklődés legegyszerűbb esete amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk.

C++ és Javaban kellett egy egyszerű programot írunk, ahol azt kellett demonstrálni, hogy a szülő osztályon csak a szülő üzeneteit lehet küldeni, a gyerek osztálynak az elemeit pedig nem.

Itt látható a kód C++-ban :

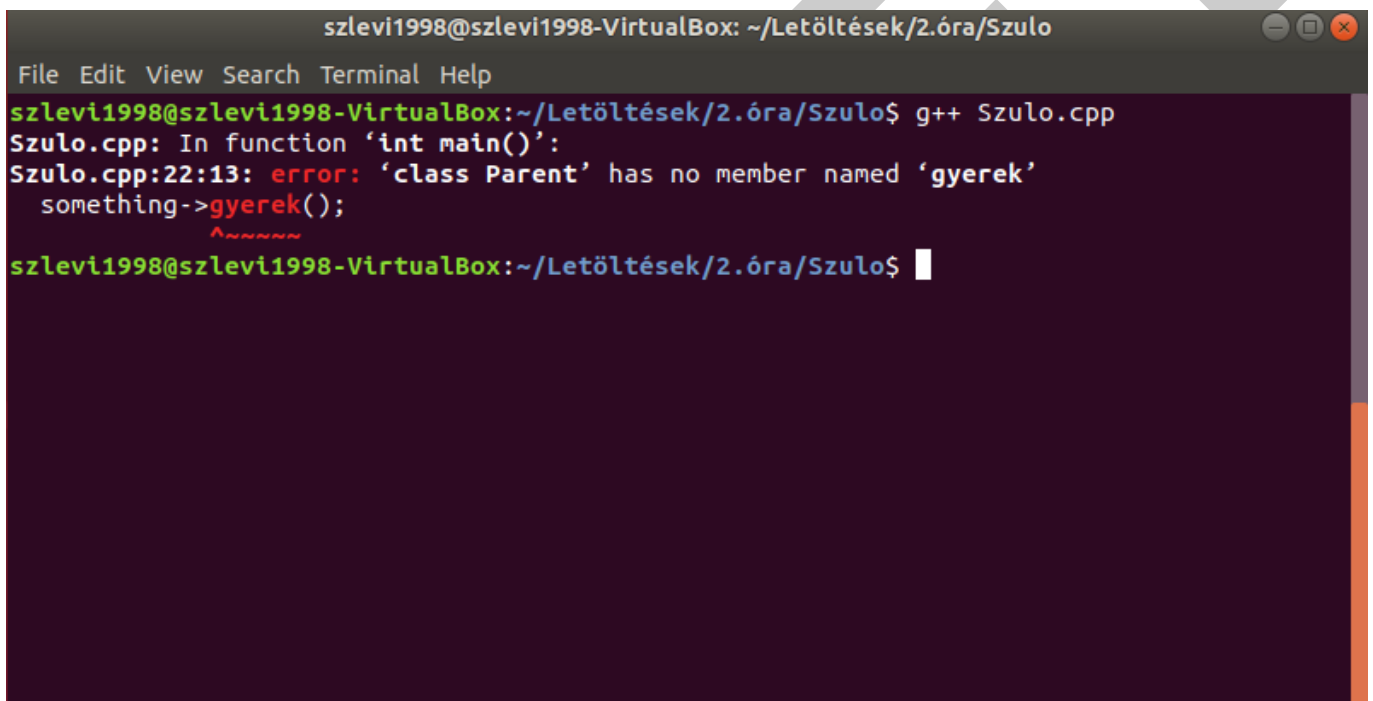
```
    ]  
#include <iostream>  
using namespace std;  
  
class Parent {  
public :  
    void parent() {  
        std::cout << " Ez a szülő osztály " << std::endl;  
    }  
};  
  
class Gyerek : public Parent {  
public :  
    void gyerek() {  
        std::cout << " Ez a gyerek osztály " << std::endl;  
    }  
};  
  
int main(){  
    Parent* something = new Gyerek();  
    something->parent();  
    something->gyerek();  
}
```

Amint látható a kódon van két osztályunk a Parent és a Gyerek osztály.

```
class Gyerek : public Parent {
```

Ebben a sorban látható az, hogy a Gyerek osztálynak átadjuk a Szülő osztályfunkcióit. A c++-ban így lehet átadni egy gyerek osztálynak a szülő osztály funkcióit.

Ezután a mainben példányosítunk és meghívjuk ezeket a függvényeket. A gyerek tudja használni a saját és a szülő osztály funkciót, azonban a szülő csak a saját funkcióit tudja alkalmazni, a gyereket viszont nem. Amint látható a program emiatt a hiba miatt nem is tud lefordulni.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'g++ Szulo.cpp' being executed. The output indicates an error in 'Szulo.cpp' at line 22, column 13: 'error: 'class Parent' has no member named 'gyerek''. The error message points to the line 'something->gyerek();'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$ g++ Szulo.cpp
Szulo.cpp: In function 'int main()':
Szulo.cpp:22:13: error: 'class Parent' has no member named 'gyerek'
  something->gyerek();
             ^~~~~~
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$
```

Itt van a kód Javában is :

```
class Parent {

    void parent() {
        System.out.println("Ez a szülő osztály");
    }
}

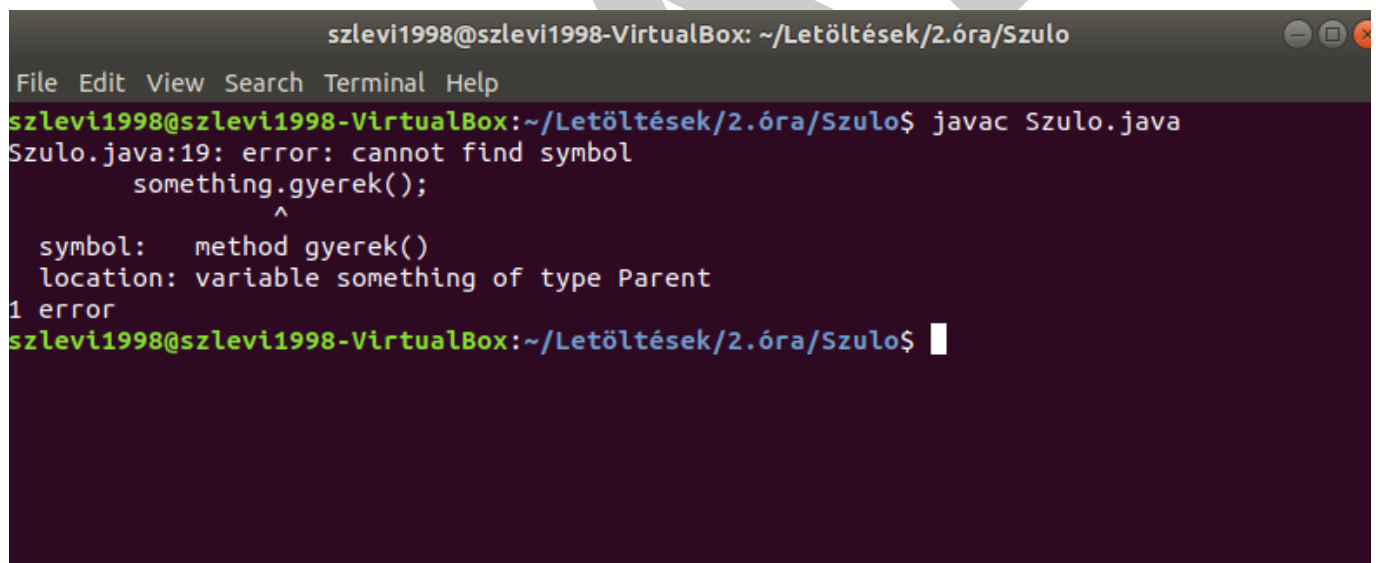
class Gyerek extends Parent {
    void gyerek() {
        System.out.println("Ez a gyerek osztály");
    }
}
```

```
class Szulo {  
    public static void main(String[] args){  
        Parent something = new Gyerek();  
  
        something.parent();  
        something.gyerek();  
    }  
}
```

Amint látható egy nagyobb különbség található meg a C++ és a Java közötti programban. A két nyelv szintaktikája alapvetően nagyon hasonlít egymásra. Azonban az öröklődés egy kicsit különbözik.

```
class Gyerek extends Parent
```

Amint látható az öröklődést itt az extends kulcsszóval lehet alkalmazni. Amint látható, nincs eltérés, hiszen a gyerek képes alkalmazni a saját és a szülő osztály funkciót, és itt is a szülő csak a saját funkciót tudja elérni, a gyereket nem.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'javac Szulo.java' being executed. The output is an error message: 'Szulo.java:19: error: cannot find symbol something.gyerek();' with an arrow pointing to the 'gyerek()' part. Below this, it says 'symbol: method gyerek()' and 'location: variable something of type Parent'. At the bottom, it says '1 error'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$ javac Szulo.java  
Szulo.java:19: error: cannot find symbol  
    something.gyerek();  
                ^  
    symbol:   method gyerek()  
    location: variable something of type Parent  
1 error  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$
```

3.3. Anti OO

Ebben a feladatban a BBP algoritmus futási időit hasonlítgatjuk össze. Sajnos az erősebb gépen a Linux nagyon sokat crashel ezért nem tudok teljes eredményt mutatni, ezért csak virtualboxos adatot tudok felmutatni. Az alábbi eredményeket kaptam:

Amint látható a Java a leggyorsabb nyelv míg a C a leglassabb. A C volt a leglassabb, ami annyira nem is meglepő, hiszen ez a nyelv a legelavultabb és mivel ez a legrégebbi, ezért nem meglepetés, az sem, hogy a legkevésbé optimalizáltabb. A Java nyert, hiszen ennek a legjobb a memória menedzselése. Érdekes volt az látni, hogy ahogy növekedett a hatvány értéke, úgy növekedett az űr a teljesítmények között.

	C	C#	Java
10 ⁶	4.168	3.788	3.614
10 ⁷	43.736	44.873	41.073
10 ⁸	509.181	487.927	456.563

3.1. táblázat. Összehasonlítás

3.4. Ciklomatikus komplexitás

Ciklomatikus komplexitás egy olyan szoftveres "mértékegység" amellyel a programunknak a komplexitását tudjuk számmal leírni. Ez a mérés a gráfelmélet alapján történik. Ezt a fogalmat McCabe komplexitásnak is nevezzük. Itt a képlete:

$$M = E - N + 2P$$

M az a szám amely a komplexitást adja meg. E az a szám amely az éleknek a számát adja abból kivonjuk a gráfnak csúcsai és hozzáadjuk az összes komponens dupláját. Én a ennek a dián(https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf) 78. oldalán lévő programkódcsipetnek a komplexitását mértem. A www.lizard.ws oldalon néztem meg az eredményét és a következő látható :

Try Lizard in Your Browser

.java
Analyse

```

@Override
protected void jatekbanVezertes(){

    if (latomAKozepet && distaceKozep < 10.0) {
        sarkonFordul();
    } else if (latomASajatGolvonalat && distanceSajatGolvonai < 0.5) {
        sarkonFordul();
    } else if (latomASzelet && distanceSzele < 5.5) {
        palyanMaradni();
    } else if (latomAFocit){
        if (distanceFoci < 0.7) {

```

Code analyzed successfully.

File Type .java
Token Count 190
NLOC 30

Function Name	NLOC	Complexity	Token #	Parameter #
jatekbanVezeries	29	15	186	

Amint láthatjuk a komplexitás ennek a kód részletnek 15. Egy jól megírt, struktúrált kód körülbelül 1 és 10 között van. 15 már egy komplex kódnak számítható, azonban nem olyan komplikált. Ez a kód még jól tesztelhető, azonban ha 20 fölötti az értékünk akkor már egy nagyon komplikált kódról beszélhetünk. 40 fölötti érték már tesztelhetlen és ilyenkor át kell gondolnunk újra a programot hiszen a kódunk túlságosan komplex.

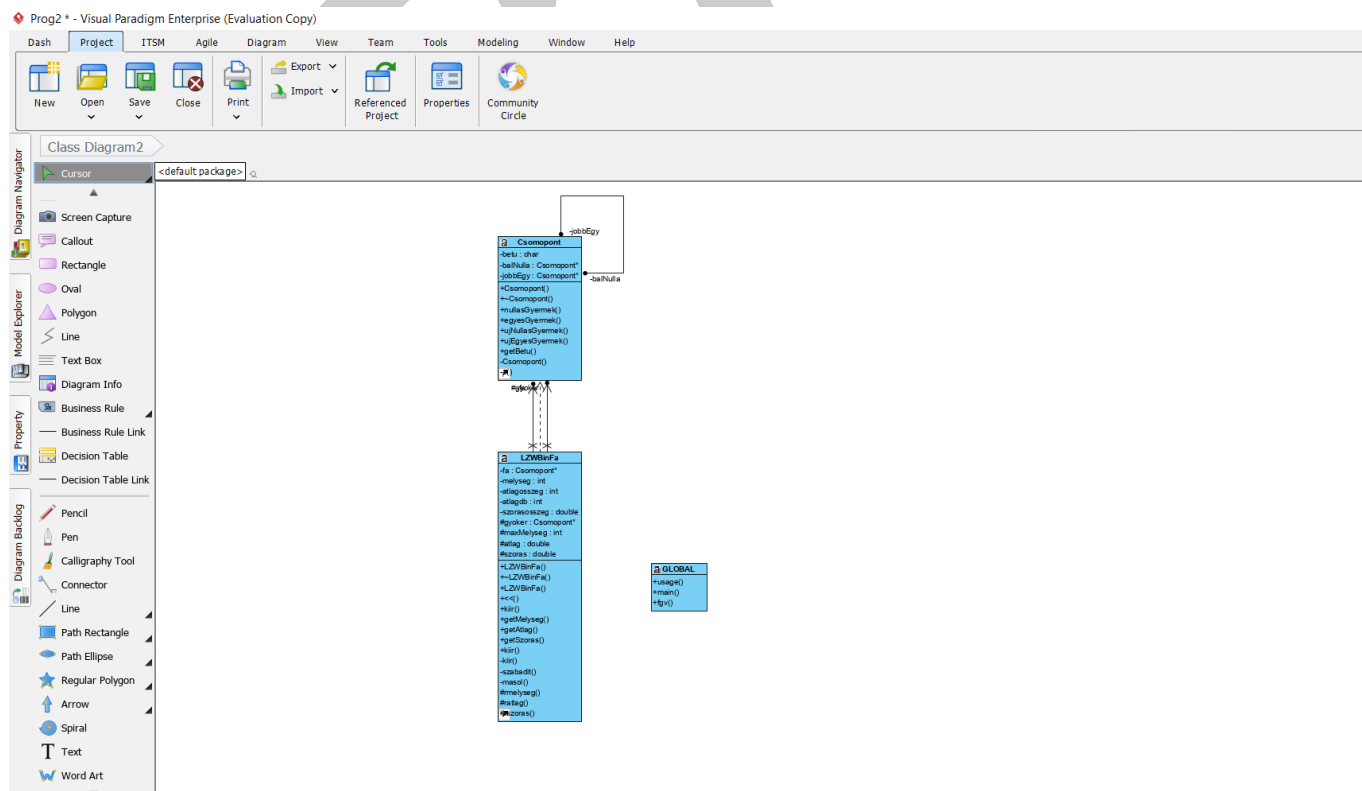
4. fejezet

Helló, Mandelbrot!

4.1. Reverse engineering UML osztálydiagram

Az UML egy szabványos, egységesített grafikus modellezőnyelv. A forráskód olyan dokumentum, ami egyértelműen definiálja a program működését. Az UML szabványos lehetőségeket kínál a rendszer felvázolásához, beleértve a fogalmi dolgokat, mint üzleti modellezés és rendszerfunkciók, valamint a konkrét dolgokat, mint programnyelvi utasítások, adatbázis sémák és újrafelhasználható szoftverkomponensek.

Ebben a feladatban az LZW binfájából kellett egy UML osztálydiagramot rajzolni. BPMN feladathoz hasonlóan itt is a visual paradigmát alkalmaztam. Bár nem tudom, hogy melyik Binfa a legmegfelelőbb ehhez a feladathoz, de én a z3a7.cpp-vel próbáltam ki. Elég egyszerű volt a feladat hiszen csak annyit kellett csinálni, hogy a tools opcióba, mint a feladatnevéből kiindulva reverselni kellett a kódot. Ezt az eredményt kaptam :



A Visual paradigm szépen lerajzolja, ezt a binfát UML verzióban. Amint látható jól elkülöníthető részeket láthatunk. Vannak Globális függvényeink, de ezek nem sok vizet zavarnak, hiszen nincsenek ágazásaiak.

Két osztályunk van a Csomópont és a LZWBInfa ezeknek vannak tagjai. Amint láthatjuk vannak jeleink (-+ #) ezek a láthatóságot jelöli. A - jel, azt jelenti, hogy a tagunk private a + a public és a # amely a protected jelzi. Csomópont osztályban van egy olyan tag, amely önmagával kapcsolja össze magát. Ezeket az összekapcsolásokat amely osztályok között vannak asszociációnak nevezzük.

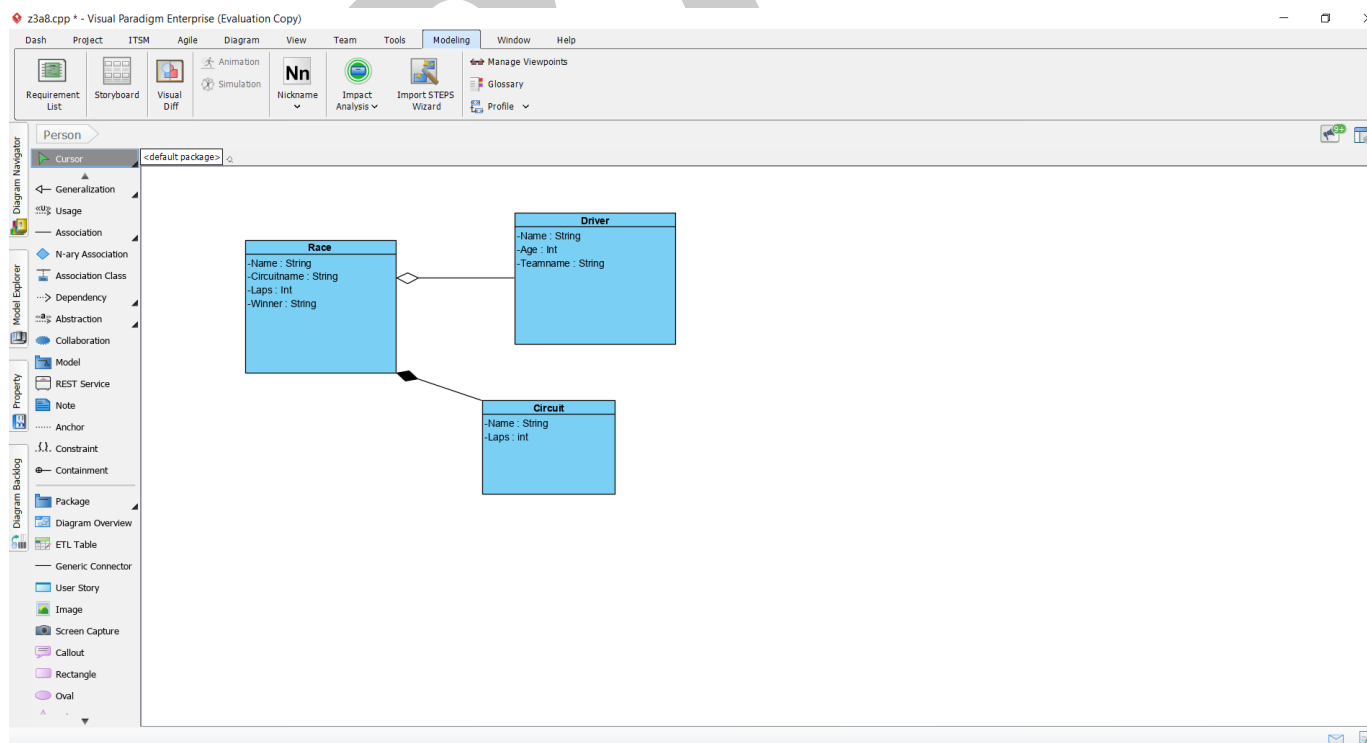
Az asszociációban az osztályok között az osztályok léte független, azonban a osztályok legtöbb esetben legalább egy osztály ismeri a másikat. Az asszociációnak két fajtája van az erős és gyenge aggregáció. A gyenge aggregáció, olyan amikor egy objektum létezhez magában is, de lehet másnak is része. Ennek a jele az UML-ben az üres rombusz.

Az erős aggregációt kompozíciónak is nevezzük, ekkor a részek élettartalma szigorúan megegyezik az egészével. A tartalmazó nem létezhet a tartalmazott nélkül. Ennek jele az UML-ben a jele a telített rombusz.

Bár az én visual paradigmmon ezeket a jelöléseket nem mutatja, mint az erős aggregációt (kompozíciót), viszont ha az egerünket rámutatjuk a kapcsolatra akkor kiírja, hogy asszociáció van a két osztály között. Láthatunk szaggatott vonalas nyilat, amely azt jelenti, hogy dependency (függőség) van az LZWBInfa és a Csomópont osztályok között. Ez azt jelenti, hogy ha valami változás lesz a Csomópont osztályban akkor kihatással lesz az LZWBInfa osztályra is.

4.2. Forward engineering UML osztálydiagram

Ebben a feladatban az előző feladatnak az ellentétjét kellett csinálni, itt UML osztályokból kellett programot készíteni. Forward engineering az a lényege, hogy egy program kódot struktúráltan, átláthatóan építsünk fel. Ennek az az előnye, hogyha egy komplikált kódot írunk, akkor gyorsan orvosolhatjuk a problémákat.



Ezen az ábrán van három osztályom, és ebből tudtam kódot generálni a Visual paradigmmeel. Ezzel a programmal és természetesen az UML-s módszerrel nagyon egyszerűen, lehet illusztrálni, hogy hol van

asszociáció, öröklődés a programunkban. Ez azért nagyon praktikus, mert ezzel sokkal gyorsabban lehet változtatni a teljes projekten. Gyakorlatilag egy nyíl "áthúzásával" nagyon sok sornyi programkódot lehet módosítani, átírni. A kisebb projekteken, mint az enyémen, annyira nem hatásos, mert nagyon kevés adatot tartalmaz, de itt is hasznos, mert ellenőrzésnek tökéletesen megfelel.

4.3. BPMN

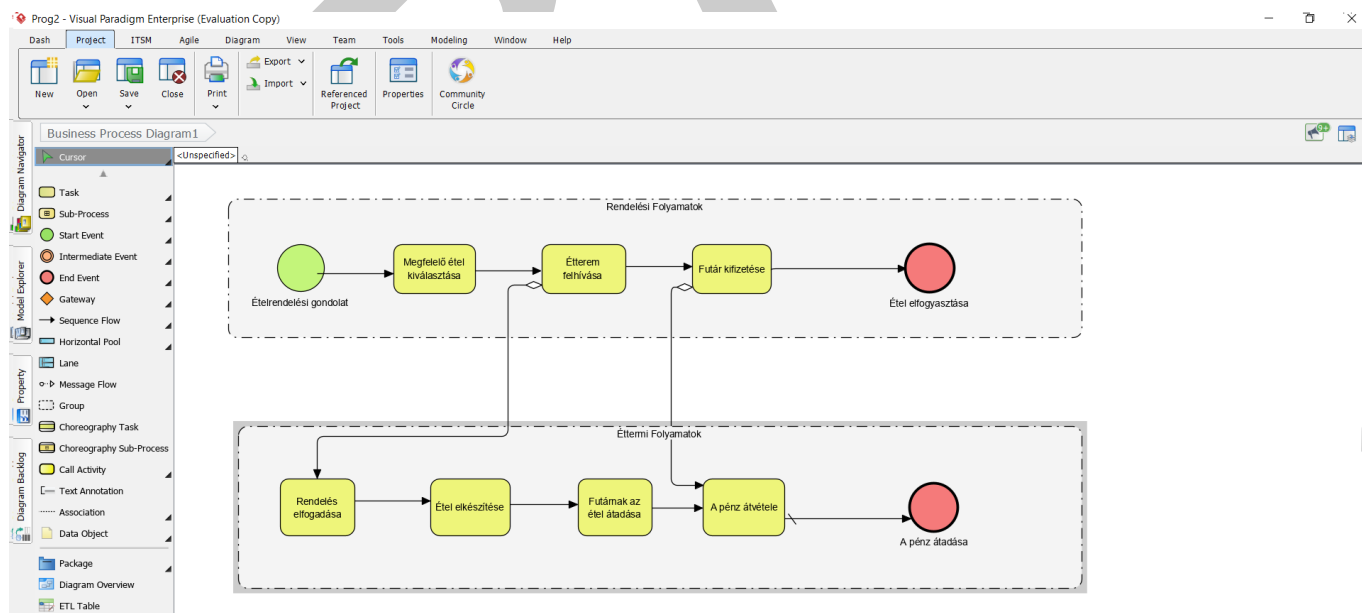
A BPMN (Business Process Modeling Notation) egy olyan folyamatábra, amely az üzleti folyamatok grafikus modellezését szolgálja. Üzleti elemzőknek és technológiai fejlesztőknek szóló grafikus jelölőnyelv.

Az üzleti folyamatok ábrázolására, bemutatására, az EPC és a BPMN módszerek a legelterjedtebbek. Az egyszerű folyamatok ábrázolásától a komplex vállalati folyamatrendszerekig képesek mindent vizualizálni. Ahhoz, hogy a valóságról kapjunk egy képet, amely alapján tudjuk elemezni majd fejleszteni a folyamatainkat elengedhetetlen, hogy azokat grafikailag ábrázoljuk.

Több fajtája is van:

- BPML (XML alapú)
- BPEL (XML alapú)
- XDPL (XML alapú)
- BPMN (Grafikus jelölésre alkalmas)

Ebbe a feladatban rajzolnunk kell egy saját folyamatábrát. Nem igazán alkalmaztam ezelőtt UML-t és ezért volt egy kisebb dilemmám, hogy milyen szoftvert alkalmazzak. Végül a Visual Paradigm-ot választottam, annak is a 30 napos próba verzióját. Egészen egyszerűen lehetett alkalmazni ezt a programot, és elkészítettem a saját folyamatomat. Íme :



Amint a képen látható egy étteremből való rendelését vezettem le. Látható egy a zöld karika, ez a kezdőpontunk. Ez a tevékenység, amely létrehozza, ezt a teljes folyamatot. Láthatunk két nagyobb halmazt, ez a két halmaz, amely a teljes folyamatot képezi. Ez a két halmaz kapcsolatba van egy mással. Láthatjuk, hogy többször is összekapcsoltam a két halmazt. Ez azért van, mert egyes tevékenységet egy adott halmaz nem tehet meg a másik "engedélye nélkül."

Láthatunk a képen sárga téglalapokat, ezek a részfolyamatok. Emelett van két piros karikánk. Ez a két piros karika amely jelzi a folyamatunknak a végét. Mivel mindkét halmaznak van egy adott teljes folyamata, ezért mindkét halmazt le kell zárni.

Természetesen ez a folyamat ábra nagyon egyszerű, a képen látható, hogy lehet belerakni gatewayeket, amelyek arra képesek hogy egy folyamatnak több elágazása legyen. Bár én nem illusztráltam a képeken de lehet olyat is csinálni, hogy beszúrok egy gatewayt az étterem felhívásnál és ha ott adok egy olyan opciót, hogy az étterem zárva, akkor a teljes folyamatnak adhatok egy végpontot.

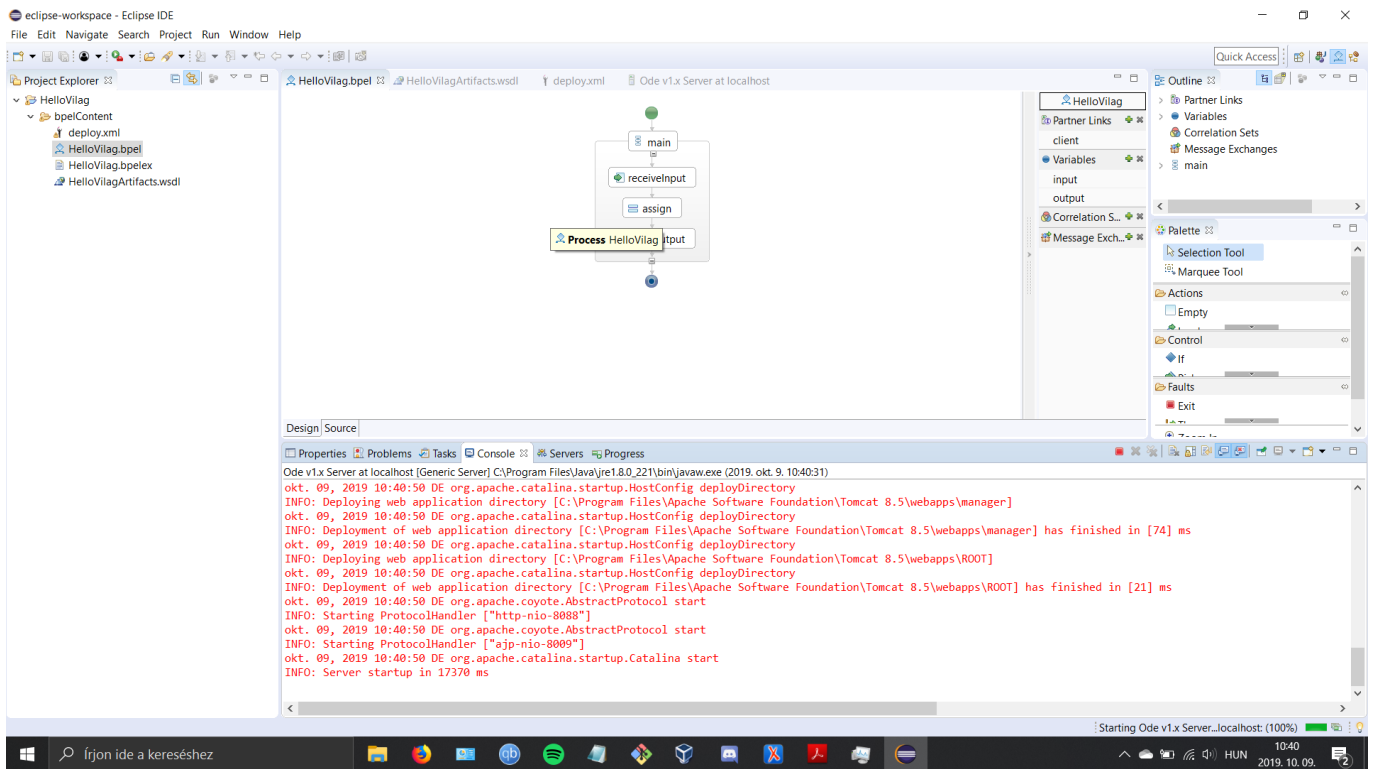
4.4. BPEL Helló, Világ! egy visszhang folyamat

BPEL (Business Process Execution Language) üzleti folyamatok modellezésének végrehajtó nyelve. XML alapú folyamatleíró nyílt szabványt alkalmaz. Elsősorban üzleti folyamatok leírására használatos, de egysegessége és elterjedtsége miatt sokszor használják munkafolyamatok leírását igénylő feladatokban is.

Minden egyes BPEL aktivitás egy külső, kiegészítő nyelven elkészített parancs meghívásával jár. A kiegészítő nyelv leggyakrabban Java, de lehet más, magas szintű script-nyelv is. A BPEL nyelvet a Microsoft és az IBM fejlesztette, illetve a korábbi BPML Üzleti folyamatokat modellező nyelvet használták fel.

Ebben a feladatban egy olyan webszervert kellett csinálni amelyben, ha egy string inputot adunk akkor a szervernek az output ugyan az a string legyen. A feladatot a Youtube-os link alapján csináltam, amely a pdf-ben található. Mivel ez a feladat "zöldes" (deprecated) eléggé nehéz volt a megtalálni a megfelelő szoftvereket találni. A legnagyobb problémám azzal adódott, hogy valamiért az Eclipse a szerverindításnál a portjaimat mindig foglaltak titálta. Végül sikerült a szervert elindítani és ezután már csak ki kellett próbálni, hogy működnek a funkciók.

Itt látható ezen a képen, hogy a szerver működik.



Valamilyen bug miatt az Eclipse-nél nem dobja fel a webservices opciót amivel tudnám a szimulálni a programomat. Ezt a hibát szeretném kijavítani később.

II. rész

Irodalomjegyzék

DRAFT

4.5. Általános

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

4.6. C

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

4.7. C++

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

4.8. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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