

Univerzális programozás

**Fekete Dávid saját programozási tankönyve
Prog2.**

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

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Fekete Dávid

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

Copyright (C) 2019, Fekete Dávid, davidfekete51@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	Bátfai, Norbert Ács Fekete, Dávid	2019. október 11.	

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
0.1.0	2019-09-19	Berners lee csokor, olvasónapló.	davidfekete
0.1.1	2019-09-26	Arroway csokor megoldása.	davidfekete

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. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Második felvonás	3
2. Helló, Berners-Lee!	5
2.1. JAVA, C++ összehasonlítás:	5
2.2. Python olvasó napló	6
3. Helló, Arroway!	7
3.1. OO szemlélet	7
3.2. Homokózó	10
3.3. „Gagyí”	10
3.4. Yoda	13
3.5. Kódolás from scratch	15
4. Helló, Liskov	16
4.1. Szülő-gyerek	16
4.2. Ciklomatikus komplexitás	18
4.3. Liskov helyettesítés sértése	19
5. Helló, Mandelbrot!	22
5.1. Reverse engineering UML osztálydiagram	22
5.2. Forward engineering UML osztálydiagram	23
5.3. BPMN	24

III. Irodalomjegyzék**26**

5.4. Általános	27
5.5. C	27
5.6. C++	27
5.7. Lisp	27

DRAFT

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 allokálni 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 más 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

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [[KERNIGHANRITCHIE](#)]
- [[BMECPP](#)]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.

II. rész

Második felvonás

DRAFT

**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

2. fejezet

Helló, Berners-Lee!

2.1. JAVA, C++ összehasonlítás:

Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven

Osztályozásnak nevezzük azt a folyamatot, amelynek során a hasonló objektumokat közös csoportokba azaz osztályokba soroljuk. Az objektumorientált programokban közös tervezésre ad lehetőséget, hogy sok objektum hasonló jellemzőkkel rendelkezik. Nagy előnyt jelent az, ha sok hasonló objektumot közös "tervrajz" alapján tudunk elkészíteni. Ezeket az ún. tervdrajzokat hívjuk osztályoknak. Az osztály bizonyos fajta objektumok közös változóit és metódusait írja le. Az osztályok definiálhatnak példányváltozókat, osztályváltozókat és osztálymetódusokat is. Az osztályváltozók az összes objektumpéldány számára megosztott információkat tartalmaznak. Ha osztályváltozókat alkalmazunk akkor feleslegesek lesznek a példányváltozók. Az osztályok legnagyobb előnye az újrafelhasználhatóság. Az objektum változókból és kapcsolódó metódusokból felépített egység. Az objektum tulajdonságait célszerű elrejtetni tehát nem publikusként kezelni és csak a metódusokon keresztül befolyásolni. Az objektumok használatának legnagyobb előnye a modularitás és az információelrejtés. Modularitás: Az objektum forráskódja független marad más objektumok forráskódjától és ennek köszönhetően könnyen tud illeszkedni a rendszer különböző részeihez. Információ elrejtés: Az objektum a publikus interfészén kommunikál a többi objektum felé. Illetve gondoskodik a saját adatairól és csak a metódusain keresztül ad változtatási lehetőséget a külső objektumoknak. A külső objektumoknak igazából nem is kell tudnia arról, hogy az objektum állapota hogyan van reprezentálva, csak a kívánt viselkedést kell kérnie a metódusokon keresztül. A példányosításhoz a new operátort használjuk a Javában. Ez egy új példányt hoz létre az osztályból és foglal helyet a memóriában. Szükségünk van egy konstruktorra is ami a hívást írja elő, ennek a neve adja majd meg, hogy melyik osztályból kell létrehozni az új példányt és inicializálja az új objektumot. A new operátor egy hivatkozást ad vissza a létrehozott objektumra. Gyakran ezt a hivatkozást hozzárendeljük egy változóhoz. Ha a hivatkozás nincs hozzárendelve változóhoz, az objektumot nem lehet majd elérni, miután a new operátort tartalmazó utasítás végrehajtódott. Az ilyen objektumot névtelen objektumnak is szoktuk nevezni. A java fordító egy bájtkódnak nevezett formátumra fordítja le a forráskódot amit a jvm önálló interpreterként fog érzékelni. Előnyös biztonsági szempontból de lassú, ezért minden jó jvm próbálja növelni a sebességet. A kódot fordítás előtt platformfüggő gépi kódra alakítja át. A nyelv szintaxisa c, c++ ból fejlődött ki, ez a szerkezetben jelenik meg de a java el is tér tőlük vagyis a hasonlóság nem egyenlő az azonossággal. C és c++-al szemben nincs alapértelmezett visszatérési érték, mindig meg kell adni, változókhöz "="-el lehet értéket rendelni kezdeti értékadás után nincs definiálva az érték. Még egy különbség, hogy név túlterhelés ugyan van a Javában

is viszont operátor túlterhelés nincs benne. Illetve számos c++ kifejezés, utasítás szintaktikailag helyes Javában is sőt sokszor a jelenetése is megegyezik. A java mint nyelv szűkebb a c++-nál, de az osztálykönyvtárai miatt szélesebb az alkalmazhatósági területe. Támogatja például a GUI programozást, network programozást vagy éppen a perzisztenciát is. C++-ban is lehet ezeket csinálni de van amelyekhez külső könyvtárak segítségét kell igénybe vennünk. Valamint lehet benne forrás szinten hordozható programokat írni de a szerkesztett bináris kód már nem hordozható, mert a lefordított kód tartalmazza a helyi oprendszerre és hardverre vonatkozó feltételezéseket. A Java egyik fő célja és egyben egyik legnagyobb különbsége a c++-hoz képest az, hogy a kód teljes mértékben hordozható. Emiatt a Java szigorúbb előírásokat szab a típusok méretére, belső szerkezetére, a kifejezések kiértékelésére és a kivételek kiváltásának ellenőrzésére. A statikus változók inicializálása is futási időben történik Javában valamint sokkal kevesebb dolgot bíz az implementációra mint a c vagy a c++. Ezt azért teszi, hogy maga a kód amely hordozható ne függjön annyira a platformtól és az implementációtól. A Java nyelv nagyon odafigyel arra, hogy a kód a lehető legpontosabban forduljon le és működjön. Ezért az ellenőrzés során kitér olyan dolgokra is amire a c++ és a c nem. Ilyen például a lokális változók ellenőrzése, pontosabban annak ellenőrzése, hogy kapnak-e értéket.

2.2. Python olvasó napló

Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)

A python programozási nyelv a C++-tól, c-től és a Javától eltérő módon arra lett inkább tervezve hogy ne a futási sebességet tegye előtérbe, hanem inkább a programozót segítse azzal, hogy könnyebben olvasható. Maga a Python egy nagyon magas szintű programozási nyelv melyet 1989 és 1991 között alkottak meg. Egy objektumorientált interpreteres nyelv (tehát rögtön futtatható, nincs különbség a forrás és a tárgykód között). Ahogy a könyv címe is sejteti az olvasóval a Python legelterjedtebb felhasználási területe a mobilprogramozás. A nyelv legfőbb jellemzője ami megkülönbözteti az általunk már jobban ismert nyelvektől és újdonság lehet a számunkra, hogy a szintaxisa behúzás alapú. Ez azt jelenti hogy az állításokat azonos szintű behúzásokkal tudjuk csoportosítani. Nem kell kapcsos zárójeleket és kulcsszavakat mint például a begin és az end használatba vennünk ehhez a feladathoz. Nagyon fontos, hogy az első utasítás a szkriptben nem lehet behúzás illetve ezeket egységesen kell kezelnünk. Továbbá az utasítások csak a sor végéig tartanak nem kell ezeket lezárni. Ha mégsem férne egy utasítás egy sorba akkor '/'-el tudjuk ezt folytatni a következő sorba.

3. fejezet

Helló, Arroway!

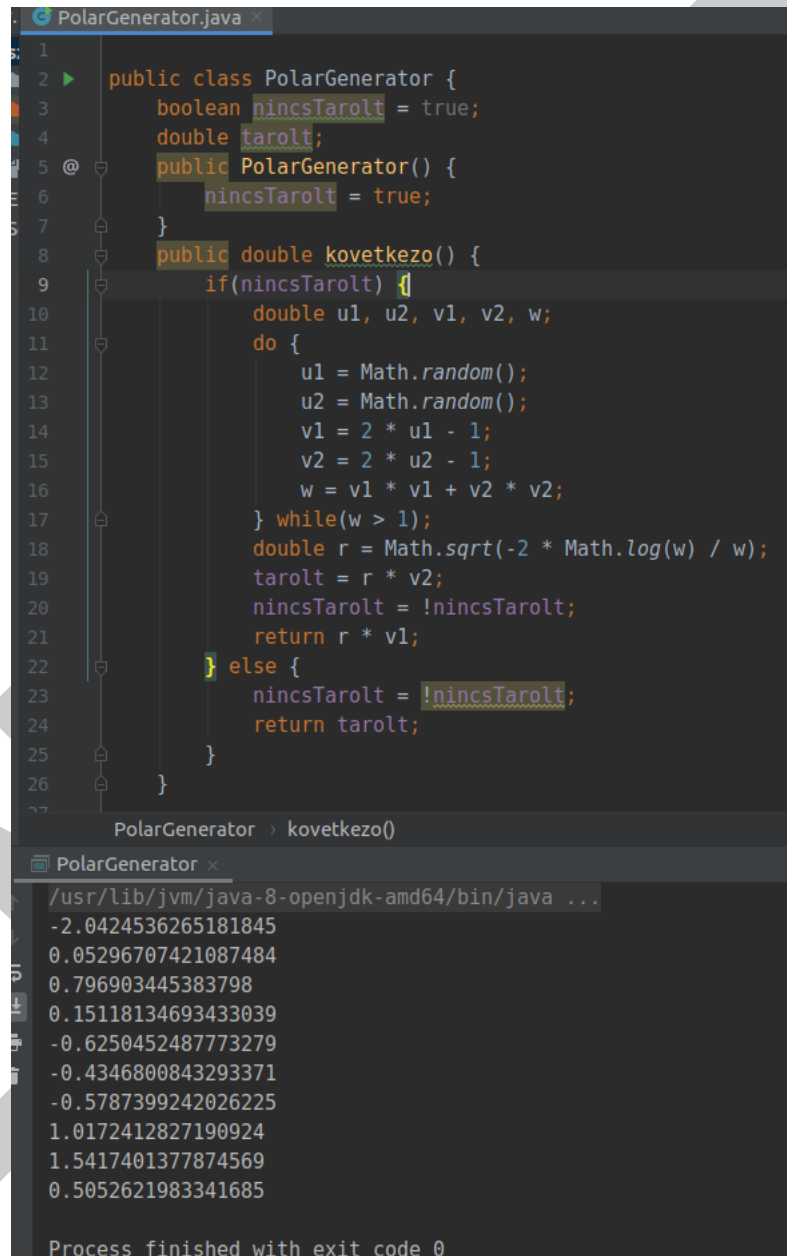
3.1. OO szemlélet

módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algoritmust.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezéséé! https://arato.inf.unideb.hu/batfai.norbert/-UDPROG/deprecated/Prog1_5.pdf (16-22 f6lia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG rep6: source/labor/polargen)

JAVA forras:

```
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());  
    }  
}  
}
```



The screenshot shows an IDE with two windows. The top window, titled 'PolarGenerator.java', displays the source code of the 'PolarGenerator' class. The code implements a method 'kovetkezo()' that generates random numbers using the Box-Muller transform. It includes a 'nincsTarolt' flag to ensure only one random number is returned per call. The bottom window, titled 'PolarGenerator', shows the output of the program, which consists of ten lines of random double values. The process finished with exit code 0.

```
1 public class PolarGenerator {  
2     boolean nincsTarolt = true;  
3     double tarolt;  
4     public PolarGenerator() {  
5         nincsTarolt = true;  
6     }  
7     public double kovetkezo() {  
8         if(nincsTarolt) {  
9             double u1, u2, v1, v2, w;  
10            do {  
11                u1 = Math.random();  
12                u2 = Math.random();  
13                v1 = 2 * u1 - 1;  
14                v2 = 2 * u2 - 1;  
15                w = v1 * v1 + v2 * v2;  
16            } while(w > 1);  
17            double r = Math.sqrt(-2 * Math.log(w) / w);  
18            tarolt = r * v2;  
19            nincsTarolt = !nincsTarolt;  
20            return r * v1;  
21        } else {  
22            nincsTarolt = !nincsTarolt;  
23            return tarolt;  
24        }  
25    }  
26 }  
27
```

PolarGenerator > kovetkezo()

```
usr/lib/jvm/java-8-openjdk-amd64/bin/java ...  
-2.0424536265181845  
0.05296707421087484  
0.796903445383798  
0.15118134693433039  
-0.6250452487773279  
-0.4346800843293371  
-0.5787399242026225  
1.0172412827190924  
1.5417401377874569  
0.5052621983341685  
  
Process finished with exit code 0
```

C++ forras:

```
#ifndef POLARGEN__H  
#define POLARGEN__H
```



```
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>

class PolarGen{
public:
    PolarGen ({
        nincsTarolt = true;
        std::srand (std::time (NULL));
    }
    ~PolarGen () {
    }
    double kovetkezo ();
private:
    bool nincsTarolt;
    double tarolt;
};
#endif
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;
    }
}

int
main (int argc, char **argv){
    PolarGen pg;
    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
    return 0;
}
```

Az objektumorientált programozás (OOP) olyan módszert nyújt a programozók számára, amely lehetővé

teszi a programok bonyolultságának csökkentését, a megbízhatóság és a hatékonyság növelését. Objektumokból, tehát a valós világ elemeinek programozási modelljeiből építi fel a programot. A C++ és a Java is objektumorientált programozási nyelv.

Remek OO bevezető példa lehet egy polártranszformációs normális generátor megírása C++-ban és Java-ban. A módosított polármódszeres algoritmus matematikai háttere a feladatmegoldás szempontjából lényegtelen, fontos viszont az a tény, hogy egy számítási lépés két normális eloszlású számot állít elő, tehát elég az előző lépés másik számát visszaadnunk.

A C++ megoldásban használjuk a scope operátort, amely lehetővé teszi, hogy hozzáférjünk az std névtérhez. Ennek köszönhetően tudunk random számot visszaadni, gyököt vonni, illetve logaritmizálni. Kiíratásnál és sortörésnél is hasznos. A nincsTárolt változóval jelöljük azt, hogy páros vagy páratlan lépésben hívtuk-e meg a ketvejelű függvényt. Ha értéke igaz, akkor tárolt lebegőpontos változóban van a visszaadandó szám.

A feladat feladata az, hogy rámutasson az objektum orientált programozás előnyeire, amire eddig konkrétan nem tértünk ki a könyvben. A program megvalósítása után azt kell látnunk, hogy nekünk, programozóknak a matematikai háttérrel alig kell foglalkoznunk, és mégis viszonylag komplex problémák megoldására vagyunk képesek programok segítségével. A kód alább látható.

A polártranszformációs normális generátor egy pszeudórandomszámok generálására kitalált algoritmus, amelynek magas effektivitást tulajdonítanak a matematikusok, hiszen nem egy, hanem kettő pszeudórandom szám jön létre egy lefutás során, hanem kettő, így minden páros lefutásnál elegendő az előzőleg generált számokból a másodikat visszaadni.

3.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiírtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját! 1

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. „Gagyí”

Az ismert formális „

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

” tesztkérdéstípusra adj a szokásosnál (miszerint 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 forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Forras:

```
import java.util.Scanner;
public class gagyi {
    public static void main(String[] args) {
        Scanner sc;
        sc = new Scanner(System.in);
        Integer x= sc.nextInt();
        Integer y= sc.nextInt();

        System.out.println(x);
        System.out.println(y);

        while (x <= y && x >= y && y != x) {
            ;
        }

    }
}
```

A feladat az, hogy választ adjunk a „

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

” tesztkérdéstípusra. A kérdésre a választ az Integer.java forrásban találjuk.

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

Az IntegerCache low értéke a -128, a high értéke pedig 127. Azaz csak abban az esetben lesz ugyanaz az objektum kiosztva mind a két értéknek, ha az -128 és 127 közé esik, vagy ezzel egyenlő. Ellenkező esetben a feltétel valamelyik ága meghívjusul, így a „return new Integer(i);” sor fog lefutni, vagyis különböző című objektumokat rendel majd az értékekhez. Írjunk java programot, amivel ezt szemléltetni tudjuk.

```
1
2 import java.util.Scanner;
3 public class gagyi {
4     public static void main(String[]args){
5         Scanner sc;
6         sc = new Scanner(System.in);
7         Integer x= sc.nextInt();
8         Integer y= sc.nextInt();
9
10        System.out.println(x);
11        System.out.println(y);
12
13        while (x <= y && x >= y && y != x) {
14            ;
15        }
16
17    }
18 }
19
gagyi > main()
```

gagyi x

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
-129 -129
-129
-129
```

Ahogy látjuk -129-es értékekre a while feltétele teljesül, vagyis végtelen ciklus jön létre.

```
1
2 import java.util.Scanner;
3 public class gagyi {
4     public static void main(String[]args){
5         Scanner sc;
6         sc = new Scanner(System.in);
7         Integer x= sc.nextInt();
8         Integer y= sc.nextInt();
9
10        System.out.println(x);
11        System.out.println(y);
12
13        while (x <= y && x >= y && y != x) {
14            ;
15        }
16
17    }
18 }
19
gagyi > main()
```

gagyi x

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
-128
-128
-128
-128
Process finished with exit code 0
```

-128-as értékekre pedig a while feltétele nem teljesül, így nem jön létre végtelen ciklus, és befejeződik a program.

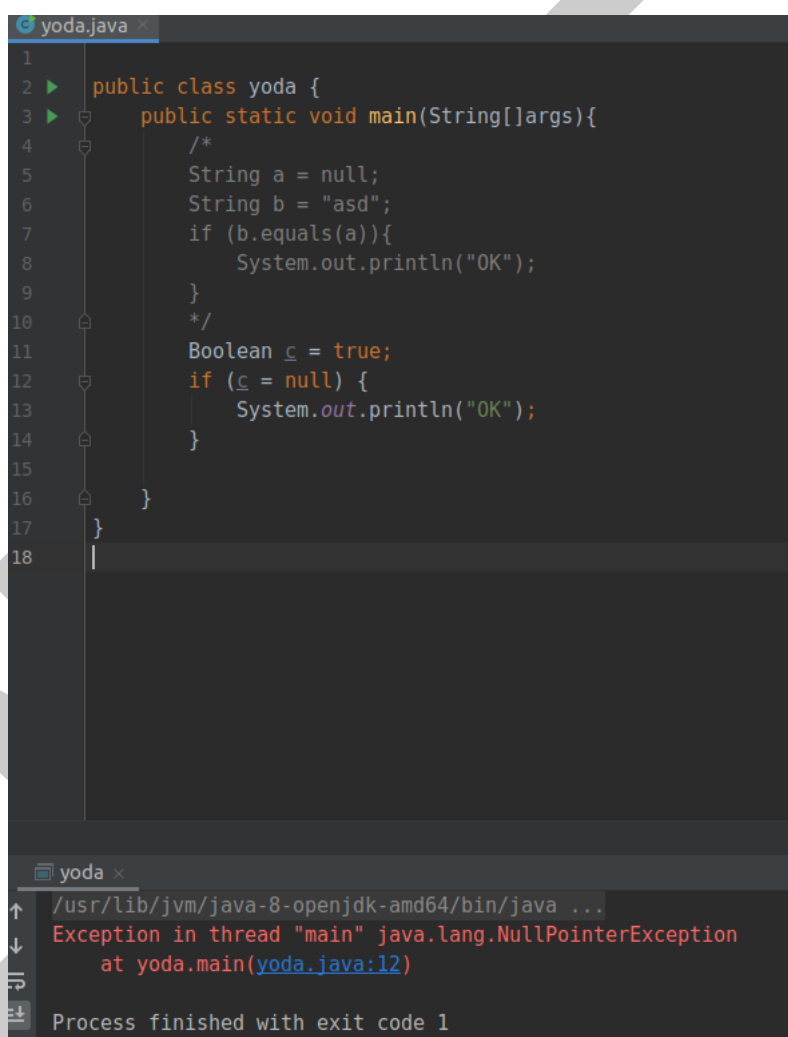
3.4. Yoda

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 programozásban a Yoda Conditions egy programozói stílus, ahol egy feltétel két része meg van cserélve egymással. A Yoda Conditions a konstans részt a feltétel bal oldalára helyezi. Ennek a programozási stílusnak a neve a Star Wars című filmből ered, amiben egy Yoda nevű karakter nem szabványos nyelvtannal beszél.

Két féle hibától is megóv a Yoda conditions: 1. ha összehasonlítás helyett értékadás történik 2. null értékű string összehasonlításnál

Írjuk meg a programot az 1. esetre, és szándékosan kövessük el a hibát:



```
1 public class yoda {
2     public static void main(String[] args){
3         /*
4          * String a = null;
5          * String b = "asd";
6          * if (b.equals(a)){
7          *     System.out.println("OK");
8          * }
9          */
10        Boolean c = true;
11        if (c = null) {
12            System.out.println("OK");
13        }
14    }
15 }
16
17
18
```

yoda x

/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...

Exception in thread "main" java.lang.NullPointerException
at yoda.main(yoda.java:12)

Process finished with exit code 1

Tehát direkt elkövetjük a hibát: összehasonlítás helyett (`==`) értékadás történik (`=`), láthatjuk, hogy a program lefordul és futtatásnál `java.lang.NullPointerException` hibát jelez.

Ha pedig követjük a Yoda conditionst, vagyis felcseréljük a feltétel két részét, a program már a fordításnál jelzi a hibát (és ez nyilván előnyösebb, mivel a fordításnál kapott hiba alapján könnyebben megtaláljuk a programkódunkban a hibát):

```
1 public class yoda {
2   public static void main(String[] args){
3     /*
4     String a = null;
5     String b = "asd";
6     if (b.equals(a)){
7       System.out.println("OK");
8     }
9     */
10    Boolean c = true;
11    if (null == c) {
12      System.out.println("OK");
13    }
14  }
15 }
```

Build x

Information: java: Errors occurred while compiling module 'Yoda'

Information: javac 1.8.0_222 was used to compile java sources

Information: 26. 9. 2019 18:43 - Build completed with 1 error and 0 warnings in 2 s 146 ms

/home/david/Yoda/src/yoda.java

Error:(12, 13) java: unexpected type
required: variable
found: value

2. eset, kövessük a Yoda conditonst: (Tulajdonképpen a 2. eset is az elsőnek egy változata: itt egy függvényhívással történik a feltétel ellenőrzése)

```
1 public class yoda {
2   public static void main(String[] args){
3
4     String a = null;
5     String b = "asd";
6     if (b.equals(a)){
7       System.out.println("OK");
8     }
9   }
10 }
```

yoda > main()

yoda x

/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...

Process finished with exit code 0

Yoda conditions követés nélkül java.lang.NullPointerException hiba:

```
1 public class yoda {
2     public static void main(String[] args){
3
4         String a = null;
5         String b = "asd";
6         if (a.equals(b)){
7             System.out.println("OK");
8         }
9     }
10 }
```

yoda > main()

yoda x

/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...

Exception in thread "main" java.lang.NullPointerException
at yoda.main(yoda.java:7)

Process finished with exit code 1

3.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp- alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Liskov

4.1. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetők! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

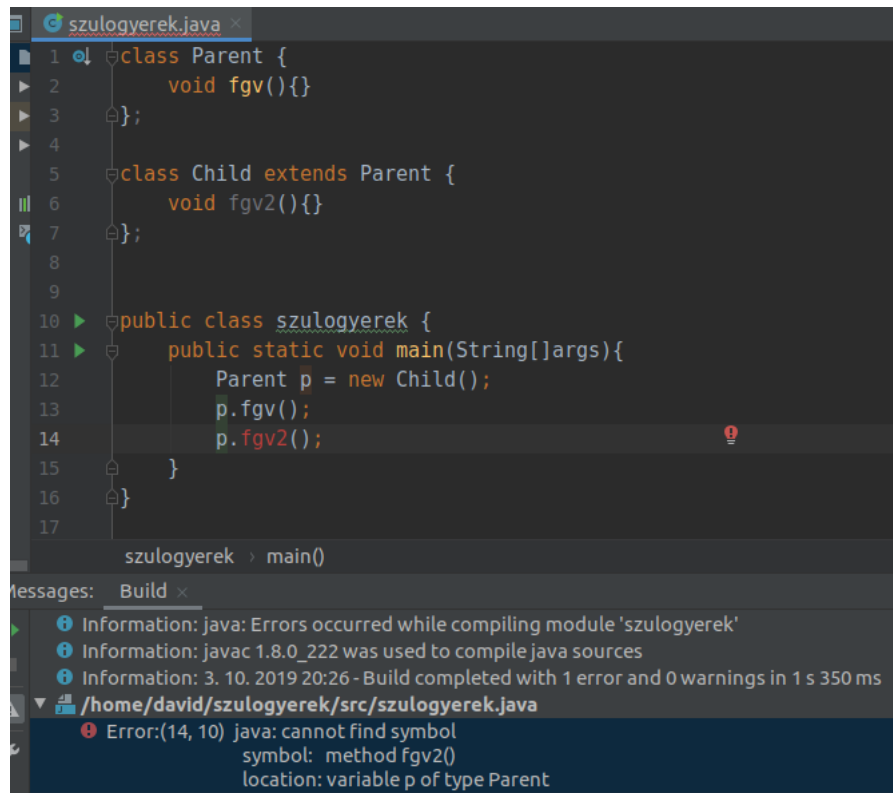
A feladat teljesítéséhez tehát írjunk Java és C++ programot, amivel bemutadjuk, hogy az ősön keresztül csak az ős üzenetei küldhetők.

Írjunk szülő-gyerek példaprogramot javaban:

```
class Parent {
    void fgv() {}
};

class Child extends Parent {
    void fgv2() {}
};

public class szulogyerek {
    public static void main(String[] args) {
        Parent p = new Child();
        p.fgv();
        p.fgv2();
    }
}
```

```
1 class Parent {
2     void fgv(){}
3 };
4
5 class Child extends Parent {
6     void fgv2(){}
7 };
8
9
10 public class szulogyerek {
11     public static void main(String[] args){
12         Parent p = new Child();
13         p.fgv();
14         p.fgv2();
15     }
16 }
17
18 szulogyerek > main()
```

Messages: Build

- Information: java: Errors occurred while compiling module 'szulogyerek'
- Information: javac 1.8.0_222 was used to compile java sources
- Information: 3. 10. 2019 20:26 - Build completed with 1 error and 0 warnings in 1 s 350 ms

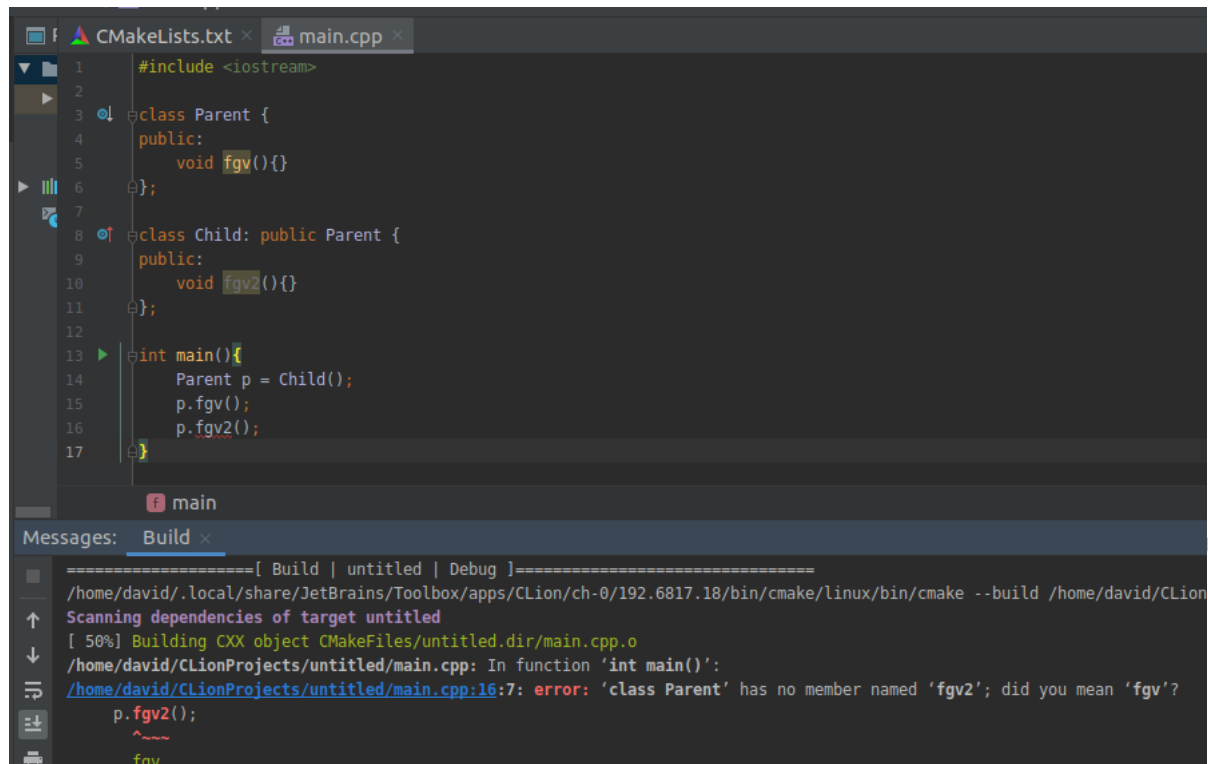
Error:(14, 10) java: cannot find symbol
symbol: method fgv2()
location: variable p of type Parent

Ugyanezt pedig c++ban is próbáljuk ki:

```
class Parent {
public:
    void fgv() {}
};

class Child: public Parent {
public:
    void fgv2() {}
};

int main() {
    Parent p = Child();
    p.fgv();
    p.fgv2();
}
```



```
1  #include <iostream>
2
3  class Parent {
4  public:
5      void fgv(){}
6  };
7
8  class Child: public Parent {
9  public:
10     void fgv2(){}
11 };
12
13 int main(){
14     Parent p = Child();
15     p.fgv();
16     p.fgv2();
17 }
```

Messages: Build ×

```
===== [ Build | untitled | Debug ] =====
/home/david/.local/share/JetBrains/Toolbox/apps/CLion/ch-0/192.6817.18/bin/cmake/linux/bin/cmake --build /home/david/CLion
[ 50%] Building CXX object CMakeFiles/untitled.dir/main.cpp.o
/home/david/CLionProjects/untitled/main.cpp: In function 'int main()':
/home/david/CLionProjects/untitled/main.cpp:16:7: error: 'class Parent' has no member named 'fgv2'; did you mean 'fgv'?
    p.fgv2();
      ^~~~~
    fgv
```

A fordításnál error kapunk mindkét esetben, mivel a Parent típusú class-on keresztül nem érjük el a Child típusú class függvényeit.

4.2. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf(77-79 fóliát)!

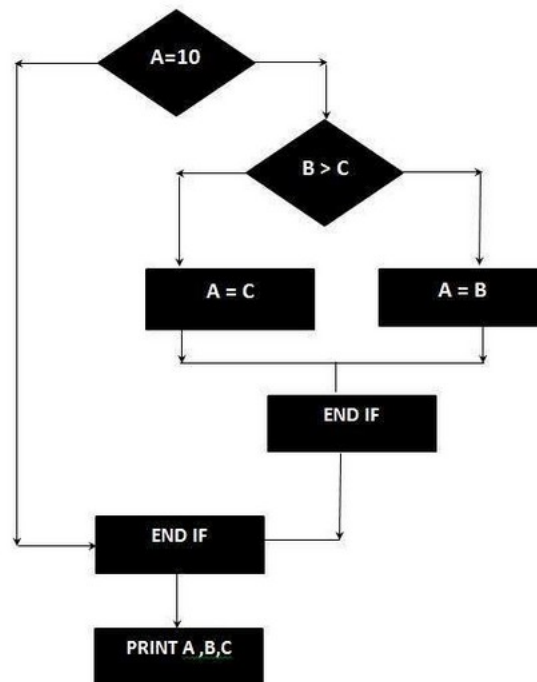
A ciklomatikus komplexitás kiszámítása: $C = E - N + 2 * P$ ahol : C = Ciklomatikus Komplexitás E = A gráf széleinek száma. N = A gráf csomópontjainak száma. P = Azon csomópontok száma, amelyeknek kilépési pontjuk van. A következő lépésben írjunk egyszerű példaprogramkódot, aminek kiszámíthatjuk a ciklomatikus komplexitását a képlet alapján.

Példaprogram c++ban:

```
#include<iostream>
using namespace std;
int main(){
    int a,b,c;
    cin >> a;
    cin >> b;
    cin >> c;
    if (a == 10){
        if (b > c){
            a = b;
        }
    }
    else {
```

```
        a = c;  
    }  
}  
cout<<a<<endl;  
cout<<b<<endl;  
cout<<c<<endl;  
}
```

Ábrázoljuk a programkódot, hogy könnyebben ki tudjuk számítani a ciklomatikus komplexitást:



Ciklomatikus komplexitás = $E - N + 2 * P$ Ebben az esetben $E = 8$, $N = 7$, $P = 1$, tehát a példaprogram ciklomatikus komplexitása 3.

4.3. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Az objektumorientált programozás öt fő tervezési elve közé tartozik az ún. Liskov-helyettesítés. Ha S osztály T osztály leszármazottja, akkor S szabadon behelyettesíthető minden olyan helyre (pl. változó), ahol T típust várunk. Tegyük fel, hogy az Allat osztály lesz a mi példánkban a T osztály. Az S osztályaink (T osztály leszármazottjai) a következők: Zebra, Oroszlan. Két osztály alkotja a P programot az LPS-ben. A programban az Zebra már nem tud vadászni, hiába lesz a leszármazott típusoknak vadász metódusa, azt a Allat allat-ra úgysem lehet hívni. Ezzel tehát a Liskov-helyettesítés elvére odafigyeltünk.

Liskovra_figyel.java :

```
#include <iostream>
using namespace std;
class Allat {
};
class RagadozoAllat: public Allat {
public:
    void vadaszik(){
        cout << "vadaszik ..." << endl;
    }
};
class Oroszlan: public RagadozoAllat {
};
class Zebra : public Allat {
};
int main ( int argc, char **argv ) {
    Oroszlan oroszlan;
    Zebra zebra;

    // zebra.vadaszik();
    oroszlan.vadaszik();
    return 0;
}
```

Példa programunkban a Madar osztály hibásan lett definiálva, mert tartalmazza a repul() metódust, így az összes Madar-ból származtatott osztály is tartalmazni fogja a repul() funkciót. Ez alapvetően hibás, mert nem minden madár tud repülni. Hibába ütközünk, amikor a Pingvin osztályt is a madárból származtatjuk, ugyanis a pingvin nem tud repülni.

```
#include <iostream>
using namespace std;
class Allat {
};
class RagadozoAllat: public Allat {
public:
    void vadaszik(){
        cout << "vadaszik ..." << endl;
    }
};
class Oroszlan: public RagadozoAllat {
};
class Zebra : public Allat {
};
int main ( int argc, char **argv ){
    Oroszlan oroszlan;
    Zebra zebra;
    //
    zebra.vadaszik();
}
```

```
oroszlan.vadaszik();  
return 0;
```

A Liskov helyettesítési elv megköveteli, hogy minden osztály legyen helyettesíthető egy gyermek osztállyal anélkül, hogy a program helyes működése megváltozna. Ebből a kódcipetből kiindulva megkezdődhet az elv megsértése. Továbbra is megmaradt a T osztály, illetve az S osztályok, viszont ezúttal nem a RagadozoAllat (S) osztályban jelenik meg a "vadászik". Ezúttal tehát így a P programban is tud vadászni az állat. Sérül a Liskov-helyettesítés elve, hiszen ebben a kódban a zebra vadászik, ami lehetetlenség.

5. fejezet

Helló, Mandelbrot!

5.1. Reverse engineering UML osztálydiagram

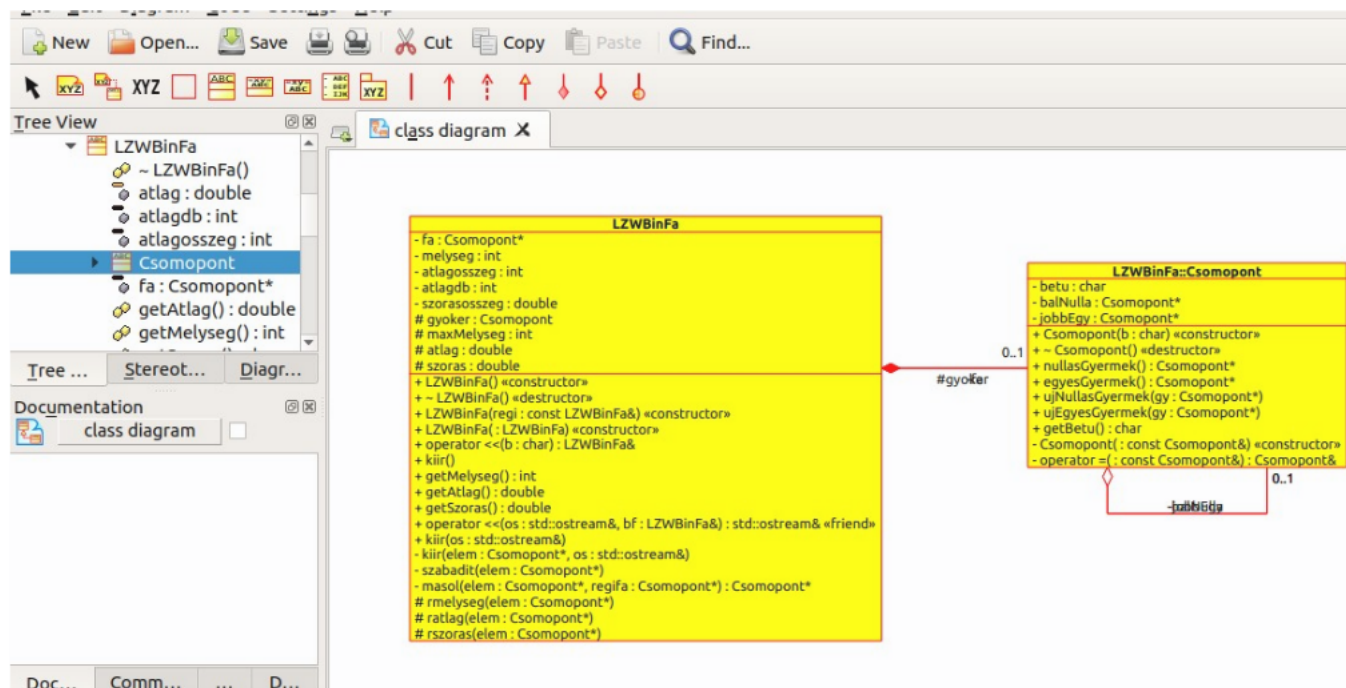
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD> (28-32 fólia)

Az UML egy egységesített modellezőnyelv, amelynek segítségével jól szemléltethetőek a fejlesztési modellek.

Kompozíció: Rész-egész kapcsolatot jelent, az egyik objektum tartalmazza vagy birtokolja a másikat.

Aggregáció: A tartalmazott a tartalmazó nélkül nem létezhet.

UML osztálydiagram létrehozására használjuk az Umbrello nevű programot. (Miután megismerkedünk az Umbrello programmal, és saját erőből létrehozunk osztálydiagramokat, egy beépített importáló eszköz segítségével programkódból UML osztálydiagramokat tudunk létrehozni, és fordítva: osztálydiagramokból programkódot generálhatunk.)

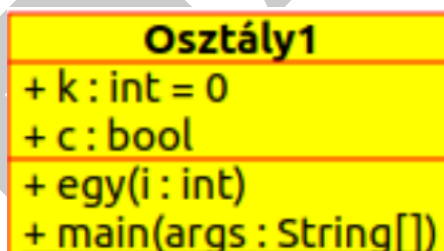


5.2. Forward engineering UML osztálydiagram

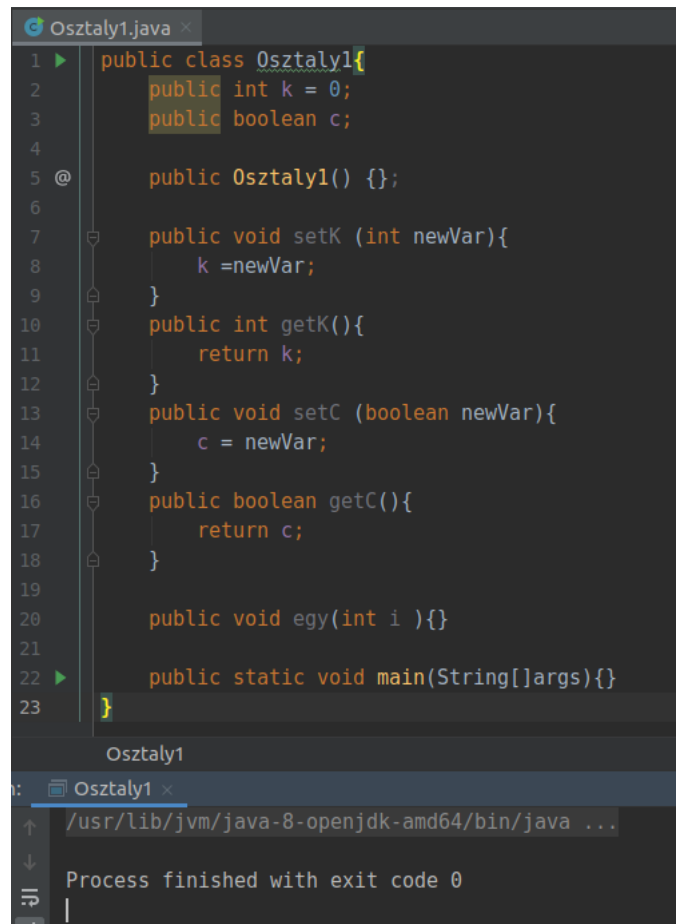
UML-ben tervezzük osztályokat és generáljuk belőle forrást!

Ehhez a feladathoz is használjuk az Umbrello nevű programot.

Az umbrello lehetővé teszi, hogy osztálydiagramokhoz különböző paramétereket vegyünk fel, amiknek kezdőértéket is adhatunk, vagy függvényeket adjunk hozzá, amihez paramétereket is beállíthatunk, illetve ezekhez kezdőértéket.



Ebből az egyszerű osztálydiagramból a következő java kódot kapjuk, amit fordíthatunk és futtathatunk is:



```
Osztaly1.java x
1 public class Osztaly1{
2     public int k = 0;
3     public boolean c;
4
5     public Osztaly1() {};
6
7     public void setK (int newVar){
8         k =newVar;
9     }
10    public int getK(){
11        return k;
12    }
13    public void setC (boolean newVar){
14        c = newVar;
15    }
16    public boolean getC(){
17        return c;
18    }
19
20    public void egy(int i ){}
21
22    public static void main(String[]args){
23    }
```

Osztaly1

Osztaly1 x

/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...

Process finished with exit code 0

5.3. BPMN

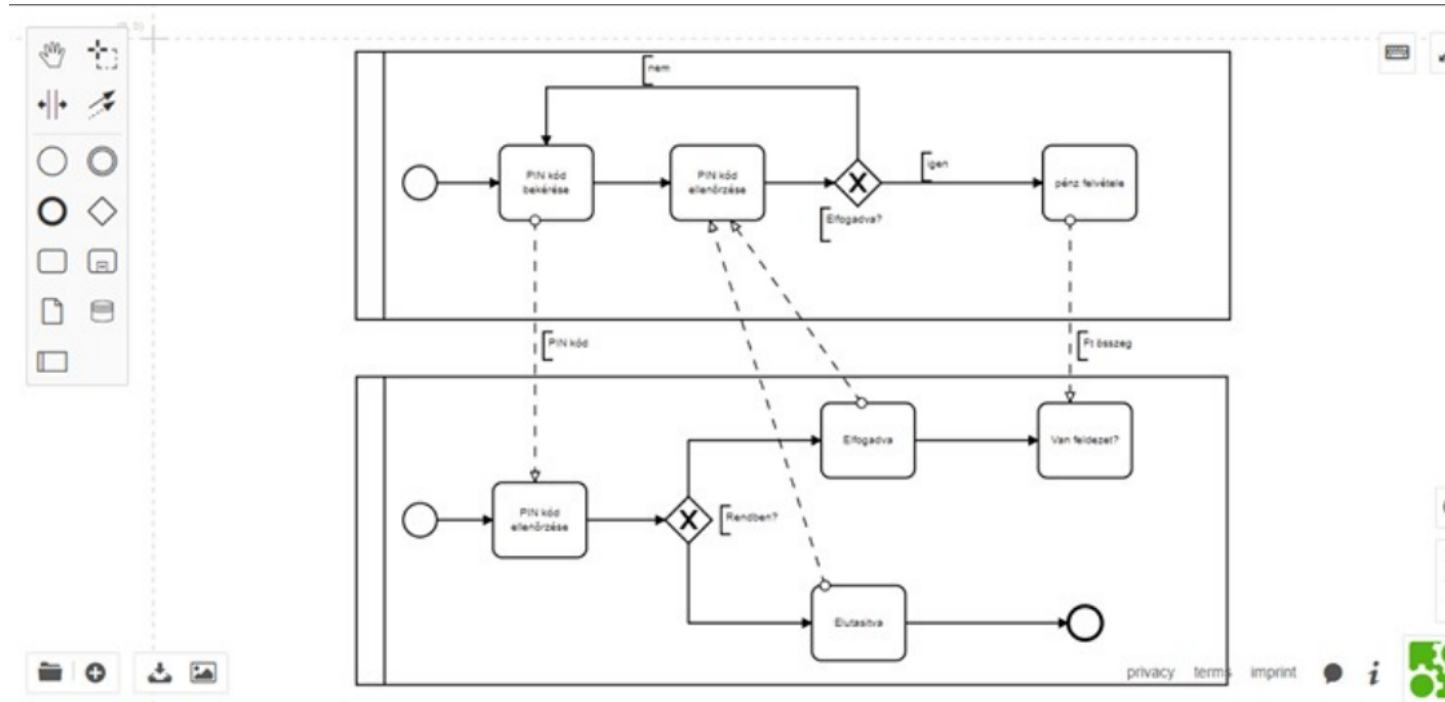
Rajzoljunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog>
(34-47 fólia)

BPMB = Business Process Model and Notation

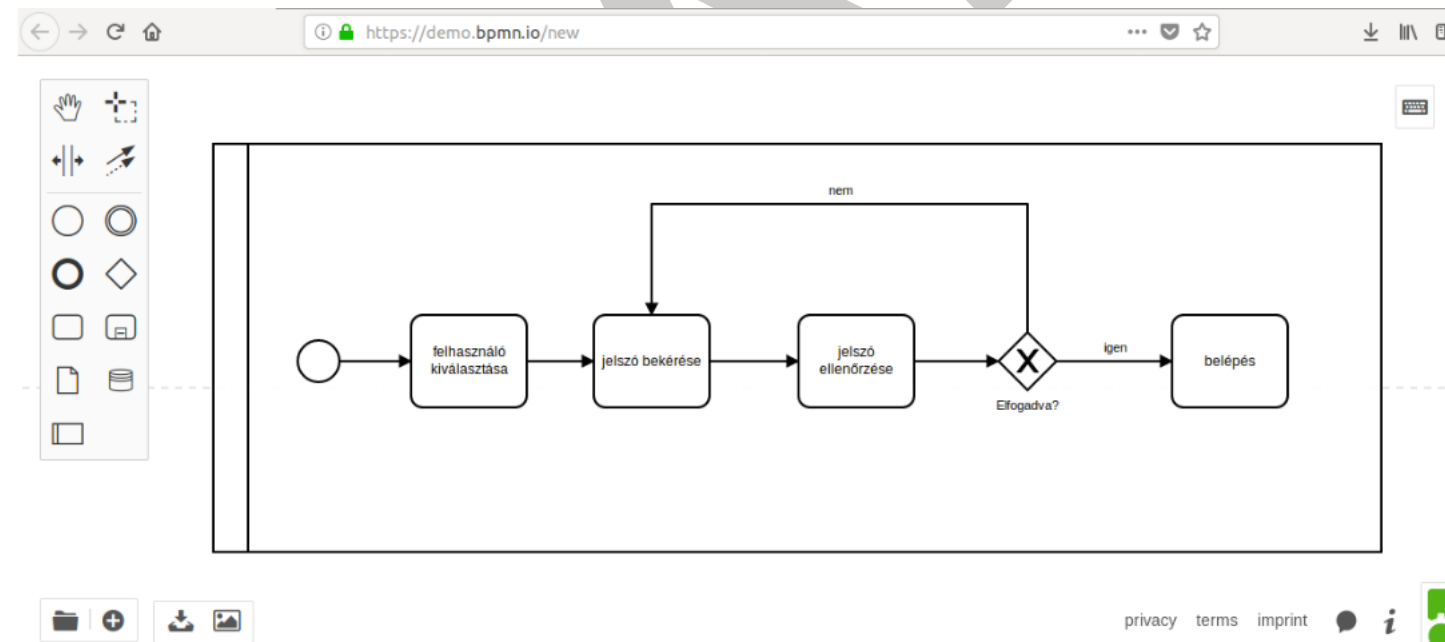
A BPMN (Business Process Model and Notation) üzleti folyamatokat reprezentál grafikusán egy üzleti folyamat modellben, amit a BPMI (Business Process Management Initiative) fejlesztett ki.

BPMN szerkesztéshez használjuk a következő online szerkesztőt: <https://demo.bpmn.io/new>

Gyakorlás képen szerkesszük meg a példában bemutatott BPMN modellt:



Ez után a feladatot teljesítve készítsünk saját modellt. (A következő BPMN modell egy operációs rendszer bejelentkezést próbál ábrázolni.)



III. rész

Irodalomjegyzék

DRAFT

5.4. Általános

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

5.5. C

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

5.6. C++

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

5.7. 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.