Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Bátfai, Norbert, Debreceni Egyetem <batfai.norbert@inf.unideb.hu>

Univerzális programozás: Írd meg a saját programozás tankönyvedet!

írta Bátfai, Norbert

kiadás dátuma 2019

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]

Köszönetnyilvánítás

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.

Tartalom

Az ábrák listája

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 (hogy lássuk mást is) példával.

Hogyan nyomjuk?

Rántsd 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 --noo
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 találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész - Bevezetés

Tartalom

1. fejezet - Vízió

Mi a programozás?

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 [https://web.archive.org/web/20181230041359if_/http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf] kódcsipeteiből is.

Milyen filmeket nézzek meg?

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

II. rész - Tematikus feladatok

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.

Tartalom

2. fejezet - Helló, Turing!

Végtelen ciklus

Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Lefagyott, nem fagyott, akkor most mi van?

Mutasd meg, hogy nem lehet olyan programot írni, amely bármely más programról eldönti, hogy le fog-e fagyni vagy sem!

Megoldás videó:

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a `Lefagy` függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne vlgtelen ciklus:

```
Program T100
{

 boolean Lefagy(Program P)
 {
   if(P-ben van végtelen ciklus)
   return true;
   else
   return false;
 }

 main(Input Q)
 {
  Lefagy(Q)
 }
}
```

A program futtatása, például akár az előző `v.c` ilyen pszeudókódjára:

```
T100(t.c.pseudo)
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra épőlő Lefagy2 már nem tartalmaz feltételezett, csak csak konkrét kódot:

```
Program T1000
{
```

```
 boolean Lefagy(Program P)
 {
   if(P-ben van végtelen ciklus)
   return true;
   else
   return false;
 }

 boolean Lefagy2(Program P)
 {
   if(Lefagy(P))
   return true;
   else
   for(;;);
 }

 main(Input Q)
 {
  Lefagy2(Q)
 }

}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés nasználata nélkül!

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/labdapattogas

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: https://youtu.be/xbYhp9G6VqQ

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R

Tanulságok, tapasztalatok, magyarázat...

3. fejezet - Helló, Chomsky!

Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használjunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A források olvasása

Hogyan olvasod, hogyan értelmezed természetes nyelven az alábbi kódcsipeteket? Például

```
if(signal(SIGINT, jelkezelo)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelo függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)

Bugok

Vigyázz, sok csipet kerülendő, mert bugokat visz a kódba! Melyek ezek és miért? Ha nem megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.
```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelo);
```

ii. `for(i=0; i<5; ++i)`

iii. `for(i=0; i<5; i++)`

iv. `for(i=0; i<5; tomb[i] = i++)`

v. `for(i=0; i<n && (*d++ = *s++); ++i)`

vi. `printf("%d %d", f(a, ++a), f(++a, a));`

vii. `printf("%d %d", f(a), a);`

viii. `printf("%d %d", f(&a), a);`

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím}})))$
```

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím}})))$
```

```
$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $
```

```
$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}}))$
```

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX

Megoldás videó: https://youtu.be/ZexiPy3ZxsA, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája
- egészek tömbje
- egészek tömbjének referenciája (nem az első elemé)
- egészre mutató mutatók tömbje
- egészre mutató mutatót visszaadó függvény
- egészre mutató mutatót visszaadó függvényre mutató mutató
- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

Mit vezetnek be a programba a következő nevek?

- `int a;`
- `int *b = &a;`
- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int (*(*z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet - Helló, Caesar!

int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

C EXOR törő

Írj egy olyan C programot, amely megtöri az első feladatban előállított titkos szövegeket!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Neurális OR, AND és EXOR kapu

R

Megoldás videó: https://youtu.be/Koyw6IH5ScQ

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R

Tanulságok, tapasztalatok, magyarázat...

Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5. fejezet - Helló, Mandelbrot!

A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

A Mandelbrot halmaz a `std::complex` osztállyal

Megoldás videó:

Megoldás forrása:

Biomorfok

Megoldás videó: https://youtu.be/IJMbgRzY76E

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf

Tanulságok, tapasztalatok, magyarázat...

A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

Mandelbrot nagyító és utazó C++ nyelven

Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteréció bejárta z_n komplex számokat!

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

Mandelbrot nagyító és utazó Java nyelven

6. fejezet - Helló, Welch!

Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

LZW

Valósítsd meg C-ben az LZW algoritmus fa-építését!

Megoldás videó:

Megoldás forrása:

Fabejárás

Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás videó:

Megoldás forrása:

Tag a gyökér

Az LZW algoritmust ültesd át egy C++ osztályba, legyen egy Tree és egy beágyazott Node osztálya. A gyökér csomópont legyen kompozícióban a fával!

Megoldás videó:

Megoldás forrása:

Mutató a gyökér

Írd át az előző forrást, hogy a gyökér csomópont ne kompozícióban, csak aggregációban legyen a fával!

Megoldás videó:

Megoldás forrása:

Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva!

Megoldás videó:

Megoldás forrása:

7. fejezet - Helló, Conway!

Hangyaszimulációk

Írj Qt C++-ban egy hangyaszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Megoldás videó: https://bhaxor.blog.hu/2018/10/10/myrmecologist

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Java életjáték

Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet - Helló, Schwarzenegger!

Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet - Helló, Chaitin!

Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome

Tanulságok, tapasztalatok, magyarázat...

Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

Tanulságok, tapasztalatok, magyarázat...

Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Omega

Megoldás videó:

Megoldás forrása:

III. 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.

Tartalom

10. fejezet - Java könyv

Olvasós

A következő bekezdésekben a Java 2 útikalauz programozóknak 5.0 első kötetének élménybeszámolóját olvashatják:

A Java programozási nyelv hasonló a C nyelvhez, azonban újabb és több lehetőséggel áll rendelkezésünkre. Nyelvünk objektumorientált, ami annyit tesz, hogy egy program objektumokra, azon belül osztályokra van felbontva, míg a C és C++ nyelvek eljárásorientáltak. Az osztályokon belül használunk változókat, illetve metódusokat. Ez utóbbi felel azért, hogy milyen műveletet hajtunk végre az adatokon. Míg a C-nél mutatókat használunk, a Java-ban már referenciákat. Változókat épp úgy adhatunk meg mint C vagy C++ nyelveknél, először a változó típusa (int, string, double, stb…), majd „=” operátor és az érték amit meg szeretnénk adni. Amikor Java kódot írunk érdemes megjegyzéseket hozzá tenni a programunkhoz (épp úgy mint bármelyik másik nyelvnél), ezt a „//” egyetlen sor esetében, ha több sorban szeretnénk megjegyzést írni akkor a „/*” nyitó részként, és a „*/” záró részként szolgáló jelöléssel tehetjük meg. Osztályokat a class kulcsszóval vezetünk be, amiken belül tetszőleges sorrendben vehetjük fel annak metódusait, és adatait. Fontos eldönteni, hogy kinek a számára legyen látható az adott elem. Így pl.: egy public osztály mindenki számára látható, míg private csak az osztályon belülre vonatkozik. Osztályon belül a new operátorral hozhatunk létre újabb objektumokat. Ügyelnünk kell a kivételkezelésre (bizonyos programok esetében pl.: nullával való osztás), ezt a try és catch segítségével oldhatjuk meg. A Java sok kétdimenziós grafikai elemmel rendelkezik, ilyen pl az awt, amivel saját magunk készíthetünk két dimenziós alakzatokat.

A legelterjedtebb karakterkészlet az ASCII kódrendszer, ami 8 biten ábrázolja a karaktereinket. Ebből azonban sok nemzet által gyakran használt karakterek hiányoznak (nálunk pl.: ű és ő betűk). Az ASCII-tól kétszer annyi biten ábrázoló Unicode azonban már rendelkezik a számunkra szükséges karakterekkel. Java-ban az azonosítók betűvel kell, hogy kezdődjenek és betűvel vagy számmal kell folytatódniuk. Tetszőleges hosszúak lehetnek, de nem tartalmazhatják a nyelv kulcsszavait (pl.: int, bolean, for, return, stb…). Az egyszerű típusokat és objektumokat literálokkal inicializáljuk, ezek pedig: objektum, logikai érték, egész szám, lebegőpontos szám, karakter, szöveg, és végül osztály. Változódeklarációnál kell hogy megadjunk egyetlen típust és legalább egy változónevet. Tömböket hasonlóan adunk meg mint C-ben, itt viszont a tömb igazi típus lesz, nem pedig mutató. Ha struktúrákat szeretnénk elérni, azokra ponttal hivatkozhatunk.

Az utasítások két fajtája: kifejezés-utasítás és deklaráció utasítás. Elágazásokat az if szerkezettel, összetettebbeket a switch-el oldhatunk meg. A Java-ban a következő ciklusok ismertek: elöltesztelő (while), hátultesztelő (do, while), léptető (for, while), és bejáró (for). Egy ciklusból a break parancs segítségével léphetünk ki, ilyenkor a program automatikusan fut tovább a következő kódsorra. Hasonlóan a continue parancs is ezt teszi, azonban ezt már használhatjuk metódusokban és inicializáló blokkokban. Visszatérési értéket C-hez hasonlóan a return parancs fog nekünk adni, viszont a Java-ból kikerült a goto utasítás, a biztonság növelése érdekében.

Java-ban az osztályok a legkisebb önálló egységek. Egy osztály egy adott tulajdonságú halmaz elemeit tartalmazza (pl.: emberek, tárgyak). Itt figyelnünk kell, hogy egy adott osztálynak csak egy célja legyen, és ne terheljük túl az osztályokat. Osztályon belüli változódeklarációnál ügyelnünk kell arra, hogy mekkora legyen a változó „hatósugara”, hogy más osztályok ne használhassák egymás változóit. Osztályon belüli metódusokat metódusdefiníciók árják le, amiknek fej és törzs része van. Metódushíváskor nem elég csak a nevet megadni, annak paramétereit is megkell. Figyelnünk kell a metódustúlterhelésre is. Ugyanis egy osztályon belül lehet több azonos nevű metódus is. Ilyenkor a fordítóprogram a paraméterek száma és típusa alapján választja ki a számunkra megfelelő metódust. Objektumokat a new operátorral hozunk létre, a new után pedig megkell adnunk, hogy melyik osztályt példányosítjuk. Példányosításkor memóriát foglalunk le, ahol az objektum változói lesznek tárolva, és visszaadja a kezdőcímét. Programozáskor problémát szokott jelenteni, hogy az objektumok feleslegesen foglalnak memóriát, mert egyszerűen nem hivatkozik rájuk semmi. Java-ban ez nem jelent problémát, mert a rendszer automatikus felszabadítja azokat helyeket, amire nincs hivatkozás. Más nyelveknél ezt a programozónak magának kell megtennie. A metódusokra nagyban hasonlítanak

a konstruktorok. Ezek végrehajtása a példányosításkor azonnal megtörténik. Egy konstruktornak meg kell egyeznie az osztály nevével. Csak példányosításon keresztül meghívhatóak. Néhány esetben nem árt értesülni egy-egy objektum megsemmisüléséről. Ez osztály szinten a classFinalize, metódus szinten simán finalize metódus (ez egy destruktor) fogja nekünk megmondani. Polimorfizmusnak nevezzük amikor egy változó olyan módon van deklarálva, hogy a leszármazottak is hivatkozhatnak rá. Polimorfizmusnál megkülönböztetünk statikus és dinamikus változókat. A statikus változó típusa változatlan, még a dinamikus változhat a változó által hivatkozott tényleges típusra.

Az osztályok mellett másik fontos építőköve a nyelvnek az interfész, ami olyan referenciatípus, amely absztrakt metódusok deklarációjának és konstans értékeknek az összege. Valódi használata az implementációján keresztül történik, így egy absztrakt program konkréttá válik. Interfészek között is van öröklődés. Deklarálni az interface kulcsszóval lehet, hasonlóan mint az osztályoknál a class. Deklarációval egy vázat hozunk létre, és ebben implementációkat helyezünk el, amik osztáylokat helyettesítenek. Fontos a fordítási hiba elkerülése végett, hogy publikus implementációkat adjunk az interfész összes metódusához. Az interfész egy újabb referenciatípus. Úgy használjuk mint egy osztályt. Az interfészek öröklődését kiterjesztésnek nevezzük, és az extend paranccsal tudjuk megvalósítani. Ez szintén úgy működik mint az osztályoknál. Konstansokat a következő képpen vezetünk be: módosítók, konstans típusa, azonosító, inicializáló kifejezés és ; zárjuk le, mint a legtöbb sorunkat. Deklarálásukhoz a módosítókat kell használjunk. Az interfészek használata az őket használó osztályok hívják meg. Ez csak akkor tud megvalósulni, ha az összes implementáló metódus szignatúrája és visszatérési értéke megegyezik az interfészével, különben error.

Amikor programozunk törekedjünk a letisztult és átlátható formára. Révén, hogy objektumorientált nyelvről van szó, a programrészeket megfelelően tagoljuk. Erre lesznek segítségünkre a csomagok, amik tartalmazzák a fejlesztői környezetet és a kódunkat. Úgy is mondhatjuk, hogy a csomag a hozzáférési kategóriák használatának az eszköze. A csomagoknál az öröklődési szint hierarchikusan van jelen. Itt a gyermeket alcsomagnak hívjuk. Tartalmuk lehetnek típusok vagy további alcsomagok, melyeket ponttal választunk el. Itt egy fa struktúrát kell elképzeljünk, ahol nincs szorosabb kapcsolat az ős felé, mint bármelyik másik csomag iránt. Célja a programkód átláthatóságának növelése, vagyis a programozó munkájának könnyítése. Csomagokat fájlrendszerben (JDK) vagy adatbázisban tárolunk. A csomagok kódja a fordítási egységekben található. Deklarálásuk package kulcsszóval történik, amik a kód legelején kell hogy legyenek, már deklaráció után nem állhatnak. Továbbá lehet importdeklarációt alkalmazni aminek három fajtája a következő: egyszerű típusimport-deklaráció, igény szerinti típusimport-deklaráció, statikus típusimport-deklaráció (ez utóbbi Java 5-től kezdve van jelen). Ezek segítik a típusok használatát. Csomagot létrehozni úgy tudunk, hogy a elkészítjük a hozzá tartozó fordítási egységeket, ezeket deklaráljuk aszerint, hogy melyik csomaghoz tartoznak. Majd megírjuk a típus~ és importdeklarációkat a fordítási egységekben. Ügyeljünk a megfelelő névválasztásra, ugyanis csoportos projekt esetén egy rosszul megválasztott csomagnév sok fejfájást okozhat. Erre egy jó módszer ha az egész elérési útját adjuk meg névként.

Sokszor esünk bele abba a hibába, hogy azt hisszük mindent jól kódoltunk le, mégis hibás a program. Azt tudjuk, hogy bizonyos részek lefutnak, na de hol keressük a bugot? Erre használjuk a kivételkezelést (Java exception), ami Java-ban a hasonlóan történik mint C-ben. Ez egy olyan különleges helyzet, hogy valahol hibát dob a program, de mégis lefut, a hibás részt pedig egy kivételobjektum fogja kezelni, pontosabb információt kapva a hibáról. Ezt nevezzük kivétel kiváltásnak (throwing exception), és szintén három fajtája fordul elő: rendellenes dolog történt (pl.: 0-val való osztás), throw utasítás kivált egy kivételt(csomagokban vagy a kódban), aszinkron hiba (párhuzamos futásnál egyik szál megszakad). A kivétel kezelése oly módon történik, hogy a kivételkezelő megkeresi azt a helyet, ahol a kiváltott kivétel kezelése megtörténhet. A kiváltott típusnak meg kell egyeznie a kivétel típusával, vagy őse az osztályhierarchiában. A kivételkezelő dönti el, hogy mekkora az a blokk amit kivált. Ezeket az utasításokat kapcsoszárójelek közé teszi. A kivételt el is kell kapnunk valahogy, erre a célra a try lesz segítségünkre. Ehhez azonban hozzá kell tennünk még közvetlenül utána egy catch vagy finally blokkot, különben errort kapunk.

A generikusok a Java 5. újdonságai, feladatuk az osztályok vagy eljárások típusokkal való paraméterezése, ez által sokkal egyszerűbb megoldani a feladatokat. Céljai az egyszerűség, biztonság növelése és a hatékonyság. Típushelyettesítő forma a wildcard ami az alábbi formák egyike lehet: ?, ? extends T, ? super T, itt T egy típust jelöl. Ügyelnünk kell, hogy a következők kivételével minden referencia-típus lehet generikus: névtelen belső osztály, kivétel típusok (oka, hogy futás közben a

generikusok nincsenek tárolva, a throwable osztály márpedig futási időben él), felsorolási típusok (mivel statikus jellegűek ugyan az a probléma mint a kivételnél). A generikusok típustörléssel foglalkoznak, és objektumok terén végrehajtott ezért object típus lesz. Figyelnünk kell az instanceof kifejezésre, ugyanis nem mindig működik generikusokra, a típustörlés következményeként.

Gyűjteményeknek nevezzük azokat a típuskonstrukciós eszközöket, melyek feladata az egy vagy több típusba tartozó objektumok példányainak memóriában történő összefoglaló jellegű tárolása, lekérdezése és manipulálása. A legtöbb programozási nyelvben metalálhatók. Általában objektumok és tömbök mutatóinak vagy referenciáinak összekapcsolása. A Java 2-ben jelent meg, inkább a praktikusság jellemző rá, mintsem a teljesség. Alapvető gyűjtemények és leképezések osztályai szorosan összefüggnek, külön-külön nem használjuk őket. Az interfészeikhez több reprezentációs osztály is tartozhat, ezek közötti váltás egyszerű. A gyűjtemény (collection) egyik leszármazottja a halmaz (set), feladata a halmaz adattípus megvalósítása. Megszorításokat tesz a, nem tesz kiterjesztést a gyűjteményhez képest. Örökli a műveleteket, amik kifejezhetők halmazműveletekként. Másik leszármazottja a lista (list), és a lista adattípust valósítja meg. Itt az elemek duplikáltan is szerepelhetnek, és számít a sorrendjük. A lista ad kiterjesztést néhány gyűjtemény interfészhez. Néhány örökölt műveletet is megváltoztat, ez duplikálás és sorrend miatt fontos (pl.: a remove művelete csak az első elemet távolítja el a listából). A sor adatszerkezet lényege, hogy az első objektumot ami bekerül, azt vegyük ki elsőnek. A leképezés adatszerkezet kulcs-érték párokat tárol, a Map interfészen keresztül. Egy kulcshoz egy érték tartozik.

Egy szoftverrendszer felépítését az alapján kell megtervezni, hogy az adatokon milyen tevékenységek lesznek elvégezve. A legáltalánosabb a felülről lefelé haladó eljárás, hiszen gyors, könnyen áttekinthető. Azonban nem minden esetben ez a legmegfelelőbb eljárás. Ezt mindig a programozónak kell eldöntenie, hogy mi lesz a legmegfelelőbb a programhoz. Objektumorientált nylevként először mindig objektumokra bontjuk a feladatot. Ez reprezentálja a modellezendő világ egységeit. Az objektumokat osztályokba soroljuk azok tulajdonságai alapján (pl.: élőlény osztálynak van ember, állat és növény objektuma). Ezen belül lesznek az alosztályok (állatoknál pl.: emlős, madár ,stb…). Amikor egy objektumot nem tudunk egyértelműen besorolni egyetlen osztályba sem, olyankor kapcsolatokkal kötjük őket osztályokhoz. Minden osztály meghatározható egy másik osztály leszűkítésével vagy kiterjesztésével. Így jön létre az örökös és gyermek kapcsolat az osztályok között. Minden gyermek osztály örökli az ősosztály tulajdonságait, fordítva nem igaz. A programtervezés legfőbb célja, hogy jól átlátható kódot készítsünk, erre az objektum orientált nyelv az egyik legjobb módszer a fentebb említett példák miatt. Szintén három lépésre oszthatjuk: analízis (a probléma körvonalazása), rendszertervezés (a körünket részekre osztjuk), osztálytervezés (a rendszert további osztályokra bontjuk). Ha mindezzel megvagyunk akkor megkezdődhet az implementálás.

A C++ jelölésrendszeréből sok mindent átvett a Java. Egy C++-ban jártas programozónak nem okoz majd nagy nehézséget megtanulni a Java szintaktikai szabályait, mert nagy átfedés van a két nyelv között, mivel a Java szintaxisa a C és C++-ból fejlődött ki. Felépítésükben viszont eltérést látunk, mivel a Java egy teljesen ″ objektumorientált nyelv. Ez annyit takar, hogy minden változó és metódus egy osztálynak a része. Amikor megírunk egy programot, aztán fordítani és futtatni szeretnénk, akkor a C ++ fordítója a saját kódunkat gépi kóddá konvertálja. A gépi kódot már értelmezni tudja az eszközünk és így a programunk már futtatható. Ennek a módszernek az a hátránya, hogy egy másik számítógépen csak azonos platform mellet lesz garantált ugyanaz az eredmény. Ezt másnéven platform- függőségnek is nevezzük. Java-ban ha a típusokról van szó, akkor primitív és nem primitív típusokra oszthatjuk fel őket. A primitív típusok egy konkrét értéket tárolnak. Ezeket a primitív típusokat helyettesíthetjük csomagoló osztályokkal is. Minden primitív adattípus rendelkezik egy hasonló (vagy vele megegyező) névű, de nagy betűs csomagoló osztállyal. Például az int-et helyettesíthetjük az Integer-rel. Osztályokat úgy mint C++-ban, Java-ban is a class teremti meg. A C++-ban különféle beépített vagy akár saját osztályok eléréséhez header fájlokat include-olása szükséges. Ez Java-ban viszont a header fájlok hiányában nem lehetséges. Helyette csomagokat használunk, amelyeket az import-tal érhetünk el, vagy simán megadhatjuk a csomag elérési útvonalát.

11. fejezet - Helló, Arroway!

OO szemlélet

Feladatunk egy prolártranszformációs normális generátor megvalósítása Java nyelven. Tekintsünk el a matematikai háttér megértéséről. Számunkra inkább maga a program működése a lényeges.

A polártranszformációs generátor Java-ban:

```
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;
        }
}
```

Az elején láthatunk egy `nincsTarolt` nevű boolean típusú váltózót. Ez fogja meghatározni a későbbi sorokban, hogy a program generál-e számot, vagy pedig az eltárolt értéket adja vissza. Ez a változó alapértéke igaz, mivel az eddig nincs semmink, amit eltárolhatnánk. Amennyiben nincs tárolt számunk, a `kovetkezo()` függvény generál nekünk két random számot, az egyiket visszaadja, a másikat pedig eltárolja egy `tarolt` nevű double-ben. Ekkor a `nincsTarolt` hamis értéket kap. Ha már rendelkezünk tárolt értékkel, akkor a függvényünk egyszerűen kiírja a tárolt számot és a `nincsTarolt` igaz értéket kap.

Ha a fenti kódot összevetjük az OpenJDK Random.java állományában találhatóval, láthatjuk, hogy lényegében megegyezik a két forráskód.

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;
 synchronized public double nextGaussian() {
        // See Knuth, ACP, Section 3.4.1 Algorithm C.
        if (haveNextNextGaussian) {
            haveNextNextGaussian = false;
            return nextNextGaussian;
        } else {
            double v1, v2, s;
            do {
                v1 = 2 * nextDouble() - 1; // between -1 and 1
                v2 = 2 * nextDouble() - 1; // between -1 and 1
                s = v1 * v1 + v2 * v2;
            } while (s >= 1 || s == 0);
            double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
            nextNextGaussian = v2 * multiplier;
            haveNextNextGaussian = true;
            return v1 * multiplier;
        }
```

Fellelhető ugyan egy apró eltérés a felcserélt if és else ágakban, de mivel `haveNextNextGaussian` alapértéke hamis, így az eredmény ugyanaz lesz.

A C++ megvalósítás is nagyban hasonlít a Java verziójához, de mint azt már tudjuk, a Java szintaktikája a C++-tól vett át sok mindent

```
#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;
    }
}
```

```
#include <iostream>
#include "polargen.h"
```

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

„Gagyi”

Az ismert formális tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj 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!

Vizsgáljuk meg a két alábbi kódcsipet eredményét.

```
public static void main (String[]args)
{
Integer x = 127;
Integer t = 127;
System.out.println (x);
System.out.println (t);
while (x <= t && x >= t && t != x);
}
```

```
public static void main (String[]args)
{
Integer x = 128;
Integer t = 128;
System.out.println (x);
System.out.println (t);
while (x <= t && x >= t && t != x);
}
```

Egy olyan ciklust szeretnénk létrehozni, ahol két számnak egyszerre kell kisebbnek (és egyenlőnek), nagyobbnak (és egyenlőnek), és nem egyenlőnek lenni. Ezt egy kis trükkel megtudjuk valósítani, ha ismerjük az `Integer` memóriafoglalási elvét. Az első példában nem következik be végtelen ciklus, mivel nem teljesültek a feltételek. Ellemben a második példában végtelen ciklusba torkollik a program, mert teljesül minden feltétel.

De hogyan teljesülhet egyszerre mind a három feltétel? A válasz abban rejlik, hogy Java-ban az `Integer` osztályban 127-ig pool-olva vannak a pozitív egészek értékei, tehát ha 127 vagy attól kisebb értéket adunk az objektumank, akkor ugyanarra a memoriórészre hivatkozik. Lényeges, hogy a != operátor nem az objektumok értékeit fogja összehasonlítani, hanem referenciákat. Mivel a 128 már nem pool-olt érték, ezért különböző memóriacímmel fog rendelkezeni a két objektum.

Yoda

Írjunk olyan Java programot, ami java.lang.NullPointerEx-el leáll, ha nem követjük a Yoda conditions-t!

A Yoda condition tulajdonképpen annyi, hogy feltétel sorrendje megfordul, és a bal oladlon fog elhelyezkedni a konstans, jobb oldalra pedig a változó kerül. Használtaával elkerülthetőek olyan hibák, mint például az ertékadó operátor(=) használata az összehasonlító helyet(==).

```
String myString = null;
if (myString.equals("foobar")) { /* ... */ }
```

Ezzel a kóddal NullPointerException-be ütközünk, mivel `null` értéket nem hasonlíthatunk konstashoz.

Nézzeük meg a Yoda conditions alakalmazásával:

```
String myString = null;
if ("foobar".equals(myString)) { /* ... */ }
```

Így viszont lefut a kódunk, mivel string konstanst hasonlíthatunk `null` értékhez. Ebben a példában hamis értéket ad a kifejezés, ahogy az várható.

Kódolás from scratch

A BBP, azazBailey-Borwein-Plouffe algoritmus segíttségével a Pi hexadecimális számjegyeit tudjuk meghatározni.

Vessünk egy pillantást a kódra.

```
public class PiBBP {
    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }

        d16PiHexaJegyek = sb.toString();
    }
```

Az algroritmus alapján a {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}} kiszámolásával kezdi a program. A {} jelölés a törtrész jelölésére szolgál. Mivel 16-os szsámrendszerben dolgozunk, és a Java nem tartalmaz hexa számjegyeket, így meg kell adnunk őket egy `hexaJegyek[]` tömbben.

```
    public double d16Sj(int d, int j) {

        double d16Sj = 0.0d;

        for(int k=0; k<=d; ++k)
            d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);
        return d16Sj - StrictMath.floor(d16Sj);
```

Itt a {16^d Sj} részt számoljuk ki. A hexajegyek kiszámolása (d+1)-edik elemtől kezdődik.

```
public long n16modk(int n, int k) {

        int t = 1;
        while(t <= n)
            t *= 2;

        long r = 1;

        while(true) {

            if(n >= t) {
                r = (16*r) % k;
                n = n - t;
            }

            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;

        }

        return r;
}
```

A 16^n mod k kiszámítása bináris hatványozással történik, ahol n a kitevő, k a modulus.

```
public String toString() {

        return d16PiHexaJegyek;
    }

    public static void main(String args[]) {
        System.out.print(new PiBBP(1000000));
    }
}
```

A `toString()` függvény a kiszámolt hexajegyek visszaadására szolgál. A `main` függvényben példányosítunk egy PiBBP objektmot. Ebben a példában d=1000000, ezért 1000001. hexajegytől írjuk ki a Pi-t.

12. fejezet - Helló, Liskov!

Liskov helyettesítés sértése

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

A Liskov elv szerint minden osztály legyen helyettesíthető leszármozattaival anélkül, hogy ez befolyásolná a program helyes működését. Tehát ha T leszármazottja S, akkor behelyettesíthetjük T helyére S-t, és ezután is helyes eredmény kell kapnunk.

Nézzünk egy példát az elvet sértő programra:

```
class Madar {
public:
     virtual void repul() {};
};
class Program {
public:
     void fgv ( Madar &madar ) {
          madar.repul();
     }
};
class Sas : public Madar
{};
class Pingvin : public Madar
{};
int main ( int argc, char **argv )
{
     Program program;
     Madar madar;
     program.fgv ( madar );
     Sas sas;
     program.fgv ( sas );
     Pingvin pingvin;
     program.fgv ( pingvin );
}
```

A Madar lett a T osztályunk. Az S osztélyok pedig a Sas és Pingvin lettek. Mivel a Madar osztály tartalmazza a repülést, így az abból származtatott Sas és Pingvin osztályok is képesek leszenk rá a porgram szerint, de tudjuk, hogy a pingvin röpképtelen, ezért ez a valóságban nem következhet be.

Nézzünk egy megoldást a madaras példára a Liskov elv betartása mellett:

```
class Madar {};
class Program {
public:
     void fgv ( Madar &madar ) {
     }
};
class RepuloMadar : public Madar {
public:
     virtual void repul() {};
};
```

```
class Sas : public RepuloMadar
{};
class Pingvin : public Madar
{};
int main ( int argc, char **argv )
{
     Program program;
     Madar madar;
     program.fgv ( madar );
     Sas sas;
     program.fgv ( sas );
     Pingvin pingvin;
     program.fgv ( pingvin );
}
```

Láthatjuk, hogy a Madar osztálynak egy új leszármazottja is van, a RepuloMadar, és ebben található a repülés. A Pingvin úgyanúgy a Madar-ból származik, azonban a Sas-t a RepuloMadar-ból származtatjuk, így csak az lesz képes a repülésre.

Végül az elv teljesülése Java-ban. A gondoltamenet ugyanaz, a szintaktika kicsit eltér.

```
class Madar {}
class Program {
 public void fgv ( Madar madar ) {}
}
class RepuloMadar extends Madar {
 public void repul() {}
}
class Sas extends RepuloMadar {}
class Pingvin extends Madar {}
public class figyel{

 public static void main ( String[] args )
 {
      Program program = new Program();
        Madar madar = new Madar();
        program.fgv(madar);

        Sas sas = new Sas();
        program.fgv(sas);
        sas.repul();
        Pingvin pingvin = new Pingvin();
        program.fgv(pingvin);
 }
}
```

Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

Ebben a feladatban be kell mutatnunk azt, hogy az ős mutatóin vagy referenciáin keresztül csak az ős metódusait érhetjük el.

Nézzünk egy rövid példakódot:

```
#include <iostream>
using namespace std;
class Szulo
{};
class Gyerek: public Szulo{
public: void kiir(){
cout<<"gyerek"<<endl;
}
};
int main()
{
Szulo* sz= new Gyerek();
cout << sz->kiir() <<endl;
}
```

A Szulo gyermekosztályaként létrehozunk egy Gyerek osztályt. Ebben a leszármazottban definiálunk egy kiir nevű eljárást. Ez a kód nem fog lefordulni helyesen, hiszen a kiir() metódus az ősosztályban nem található meg.

Ha ezt a kódot Java-ban próbáljuk megvalósítani szintén problémába ütközünk, ugyanis nem hivható meg a leszármazott metódusa az ősön keresztül. Errort kapunk fordításnál, jogosan. Íme a kód Java-ban:

```
public class Szulo{
 public class Gyerek extends Szulo{
  public void kiir(){System.out.println("Gyerek");}
 }
 public  void main(String[] args)
 {
  Szulo szulo = new Gyerek();
  System.out.println(szulo.kiir());
 }
}
```

Ciklomatikus komplexitás

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

A ciklomatikus komplexitás vagy másnéven McCabe-komplexitás (alkotója után kapta ezt a nevet) a program forrásának komplexitását határozza meg egy konkrét számértékkel. A számítása gráfelméletre alapul. A képlet így néz ki: `M = E - N + 2P` , amelyben E a gráf éleinek száma, N a csúcsok száma és P az összefüggő komponensek száma.

Akkor nézzük is meg egy tetszőleges forráskód ciklomatikus komplexitás. Ez most az előzőekben már többször használt PiBBP algoritmus Java verziója lesz. Én a forrás ciklomatikus komplexitását a: Lizard [http://www.lizard.ws/] nevű open source elemzővel számoltattam ki. Tetszőleges kódot másolhatunk be a bal oldalra, majd kiválasztjuk a programnyelvet, és futtatjuk rajta az elemzőt. Az eredmény ta jobb oldalon kapjuk meg:

12.1. ábra - PiBBP.java ciklomatikus komplexitása

Function Name	NLOC	Complexity	Token #	Parameter #
PiBBP::PiBBP	20	3	192	
PiBBP::d16Sj	6	2	70	
PiBBP::n16modk	17	5	87	
PiBBP::toString	3	1	8	
PiBBP::main	3	1	22	

EPAM: Interfész evolúció Java-ban

Mutasd be milyen változások történtek Java 7 és Java 8 között az interfészekben. Miért volt erre szükség, milyen problémát vezetett ez be?

Vizsgáljuk meg az alábbi két kódot:

```
            package com.epam.training;

public interface InterfaceA {

 default public void method() {
  System.out.println("InterfaceA.method()");
 }

}
```

A Java 7, vagy korábbi verziókban nem lehetett implementálni az interfészekben default metódust. Ehhez képest a Java 8-ban újításnak hatott default metódusra azért volt szükség, hogy könnyebb legyen biztosítani a visszafele kompatibilitást az ebben a verzióban megjelent újításokkal kapcsolatban. Lássuk milyen problémát vetett is ez fel:

```
            package com.epam.training;

public interface InterfaceB {

 default public void method() {
  System.out.println("InterfaceB.method()");
 }

}
```

A példákban jól látható, hogy ha több interfész is rendelkezik ugyan azzal a default metódussal (és adnak hozzá default implementációt), akkor fellép egy többszörös öröklődési probléma. Erre kénytelenek vagyunk megadni egy explicit módon felüldefiniáló osztályt, ahol implementáljuk a két interfészt, és eldöntjük, hogy még is melyiket szeretnénk használni.

```
public class Implementation implements InterfaceA, InterfaceB {

 @Override
 public void method() {
  InterfaceA.super.method();
 }

}
```

Irodalomjegyzék

Általános

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

C

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

C++

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

Lisp

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