

# **Magas szintű programozási nyelvek II.**

---

## **Labor jegyzőkönyv**

Ed. PROG2, DEBRECEN,  
2020. szeptember 8, v. 0.0.1

## COLLABORATORS

	<i>TITLE :</i> Magas szintű programozási nyelvek II.		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Dude, Dude	2020. szeptember 9.	

## REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2020-09-08	Inited	rkeeves

## Ajánlás

DRAFT

# Tartalomjegyzék

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Labor jegyzőkönyv</b>	<b>2</b>
1.1. Mi a cél? . . . . .	2
<b>II. Tematikus feladatok</b>	<b>3</b>
<b>2. Helló, Berners-Lee!</b>	<b>5</b>
2.1. C++ Java összehasonlítás . . . . .	5
2.2. Python Java Mobil . . . . .	5
<b>3. Helló, Arroway!</b>	<b>6</b>
3.1. OO szemlélet . . . . .	6
3.2. Homokozó . . . . .	12
3.3. Gagyi . . . . .	18
3.4. Yoda . . . . .	18
3.5. Kódolás from scratch . . . . .	19
3.6. EPAM: Java Object metódusok . . . . .	19
3.7. EPAM: Eljárásorientál vs Objektumorientált . . . . .	19
3.8. EPAM: Objektum példányosítás programozási mintákkal . . . . .	19
<b>III. Irodalomjegyzék</b>	<b>20</b>
3.9. Általános . . . . .	21
3.10. My Little Ponys . . . . .	21

## Ábrák jegyzéke

3.1. Java Manual Build	8
3.2. Manual Build	10
3.3. Make Build	12
3.4. Lzw Java	18

# Előszó

...

## Használat

Használj Linuxot. Én múlt félévben Cygwinnel és Mingw-vel trükköztem. Meg lehet csinálni de kétszer annyi idő lesz.

Főleg Ubuntun és Debian-on könnyen felmegy apt-tal a dblatex. Amit ne felejts el, az a hungarian language pack. texlive-lang-hungarian vagy texlive-lang-european.

## **I. rész**

### **Bevezetés**

DRAFT

# 1. fejezet

## Labor jegyzőkönyv

### 1.1. Mi a cél?

A laborjegyzőkönyvnek a saját munka dokumentálása a célja.

DRAFT



## **II. rész**

### **Tematikus feladatok**

DRAFT

**Bátf41 Haxor Stream**

Erősen javaslom ezen jegyzőkönyv olvasása helyett [Bátfai Tanár Úr csatornáját](#). [Perma YT archive](#).

---

DRAFT

---

## 2. fejezet

# Helló, Berners-Lee!

### 2.1. C++ Java összehasonlítás

Ebből a két könyvből pár oldalas esszé jellegű kidolgozást kérek, Java és C++ összehasonlítás mentén, pl. kb.: kifejezés fogalom ua., Javában minden objektum referencia, mindig dinamikus a kötés, minden függvény virtuális, klónozásstb.

Tanulságok, tapasztalatok, magyarázat...

### 2.2. Python Java Mobil

Itt a kijelölt oldalakból (35-51) egy 1 oldalas élmény-olvasónaplóra gondoltam.

Tanulságok, tapasztalatok, magyarázat...

## 3. fejezet

# Helló, Arroway!

### 3.1. OO szemlélet

A 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 algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!Lásd még fóliák!Ismétlés: [\(16-22 fólia\)](#) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: [source/labor/polargen](#))

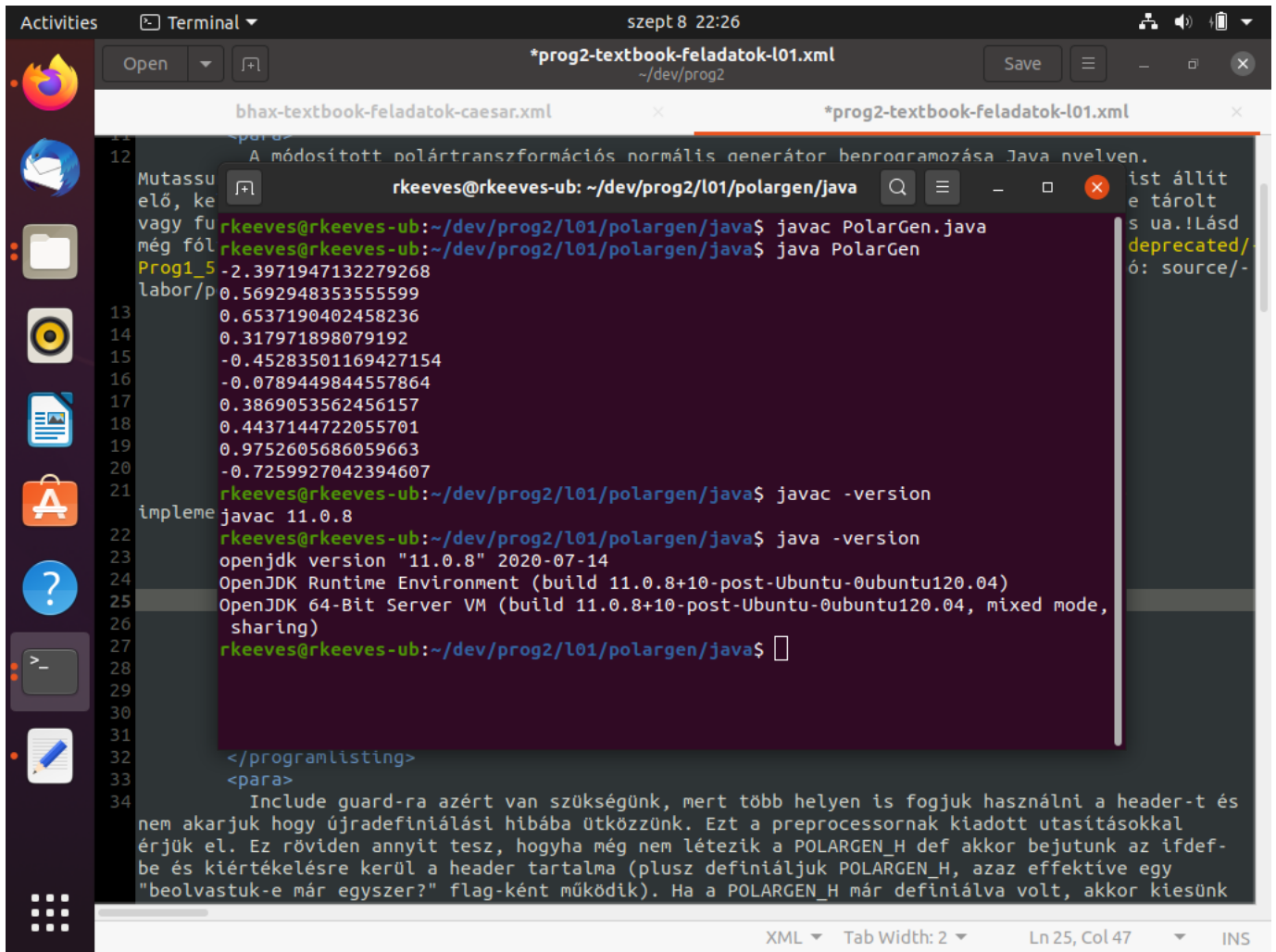
A Java implementáció a következő:

```
1 import java.lang.Math;
2
3 public class PolarGen
4 {
5     boolean store_is_empty;
6
7     double stored;
8
9     public PolarGen()
10    {
11        store_is_empty = true;
12    }
13
14    public double next()
15    {
16        if(store_is_empty){
17            double u1, u2, v1, v2, w;
18            do{
19                u1 = Math.random();
20                u2 = Math.random();
21                v1 = 2 * u1 - 1;
22                v2 = 2 * u2 - 1;
23                w = v1 * v1 + v2 * v2;
24            }while(w > 1);
25            double r = Math.sqrt((-2 * Math.log(w)) / w);
```

```
26     stored = r * v2;
27     store_is_empty = !store_is_empty;
28     return r * v1;
29 }else{
30     store_is_empty = !store_is_empty;
31     return stored;
32 }
33 }
34
35
36 public static void main(String[] args){
37     PolarGen g = new PolarGen();
38     for(int i = 0; i < 10; ++i){
39         System.out.println(g.next());
40     }
41 }
42 }
```

Látható hogy egy fordítási egységen belül definiáltunk egy PolarGen osztályt és implementáltuk is.

Ezen kicsi fájl miatt nem érdemes elindítani az IDE-t. Alább egy fordítás és futtatás OpenJDK 11.0-val, mivel az Oracle volt olyan kedves hogy ... na ebbe most ne menjünk bele-



```
Activities Terminal szept 8 22:26
*prog2-textbook-feladatok-l01.xml
~/dev/prog2
bhax-textbook-feladatok-caesar.xml *prog2-textbook-feladatok-l01.xml
12 Mutassu A módosított polártranszformációs normális generátor beprogramozása Java nvelven.
elő, ke rkeeves@rkeeves-ub: ~/dev/prog2/l01/polargen/java
vagy fu rkeeves@rkeeves-ub:~/dev/prog2/l01/polargen/java$ javac PolarGen.java
még föl rkeeves@rkeeves-ub:~/dev/prog2/l01/polargen/java$ java PolarGen
Prog1_5 -2.3971947132279268
labor/p 0.5692948353555599
13 0.6537190402458236
14 0.317971898079192
15 -0.45283501169427154
16 -0.0789449844557864
17 0.3869053562456157
18 0.4437144722055701
19 0.9752605686059663
20 -0.7259927042394607
21 rkeeves@rkeeves-ub:~/dev/prog2/l01/polargen/java$ javac -version
javac 11.0.8
22 rkeeves@rkeeves-ub:~/dev/prog2/l01/polargen/java$ java -version
openjdk version "11.0.8" 2020-07-14
23 OpenJDK Runtime Environment (build 11.0.8+10-post-Ubuntu-0ubuntu120.04)
24 OpenJDK 64-Bit Server VM (build 11.0.8+10-post-Ubuntu-0ubuntu120.04, mixed mode,
25 sharing)
26 rkeeves@rkeeves-ub:~/dev/prog2/l01/polargen/java$
27
28
29
30
31
32 </programlisting>
33 <para>
34 Include guard-ra azért van szükségünk, mert több helyen is fogjuk használni a header-t és
nem akarjuk hogy újradefiniálási hibába ütközzünk. Ezt a preprocessornak kiadott utasításokkal
érjük el. Ez röviden annyit tesz, hogyha még nem létezik a POLARGEN_H def akkor bejutunk az ifdef-
be és kiértékelésre kerül a header tartalma (plusz definiáljuk POLARGEN_H, azaz effektíve egy
"beolvastuk-e már egyszer?" flag-ként működik). Ha a POLARGEN_H már definiálva volt, akkor kiesünk
```

3.1. ábra. Java Manual Build

A Cpp esetén egy header-ben definiáltuk az osztályt.

```
1 #ifndef POLARGEN_H
2 #define POLARGEN_H
3
4 namespace prog2{
5     class PolarGen
6     {
7
8     public:
9         PolarGen();
10
11         ~PolarGen() = default;
12
13         double next();
14
15     private:
16
```

```
17     bool store_empty;  
18  
19     double stored;  
20  
21 };  
22 } /* prog2 */  
23 #endif /* POLARGEN_H */
```

Include guard-ra azért van szükségünk, mert több helyen is fogjuk használni a header-t és nem akarjuk hogy újradefiniálási hibába ütközzünk. Ezt a preprocessornak kiadott utasításokkal érjük el. Ez röviden annyit tesz, hogyha még nem létezik a POLARGEN\_H def akkor bejutunk az ifdef-be és kiértékelésre kerül a header tartalma (plusz definiáljuk POLARGEN\_H, azaz effektíve egy "beolvastuk-e már egyszer?" flag-ként működik). Ha a POLARGEN\_H már definiálva volt, akkor kiesünk ifdefből és végeztünk.

Eltértem Tanár Úr példájától, hisz nem láttam értelmét a header-be includeolni a rand-hoz szükséges dolgokat. Ezek elegendőek ha bekerülnek a cpp-be. Ehhez csak annyit kell módosítanunk hogy ctor implementációt a cpp-ben végezzük el.

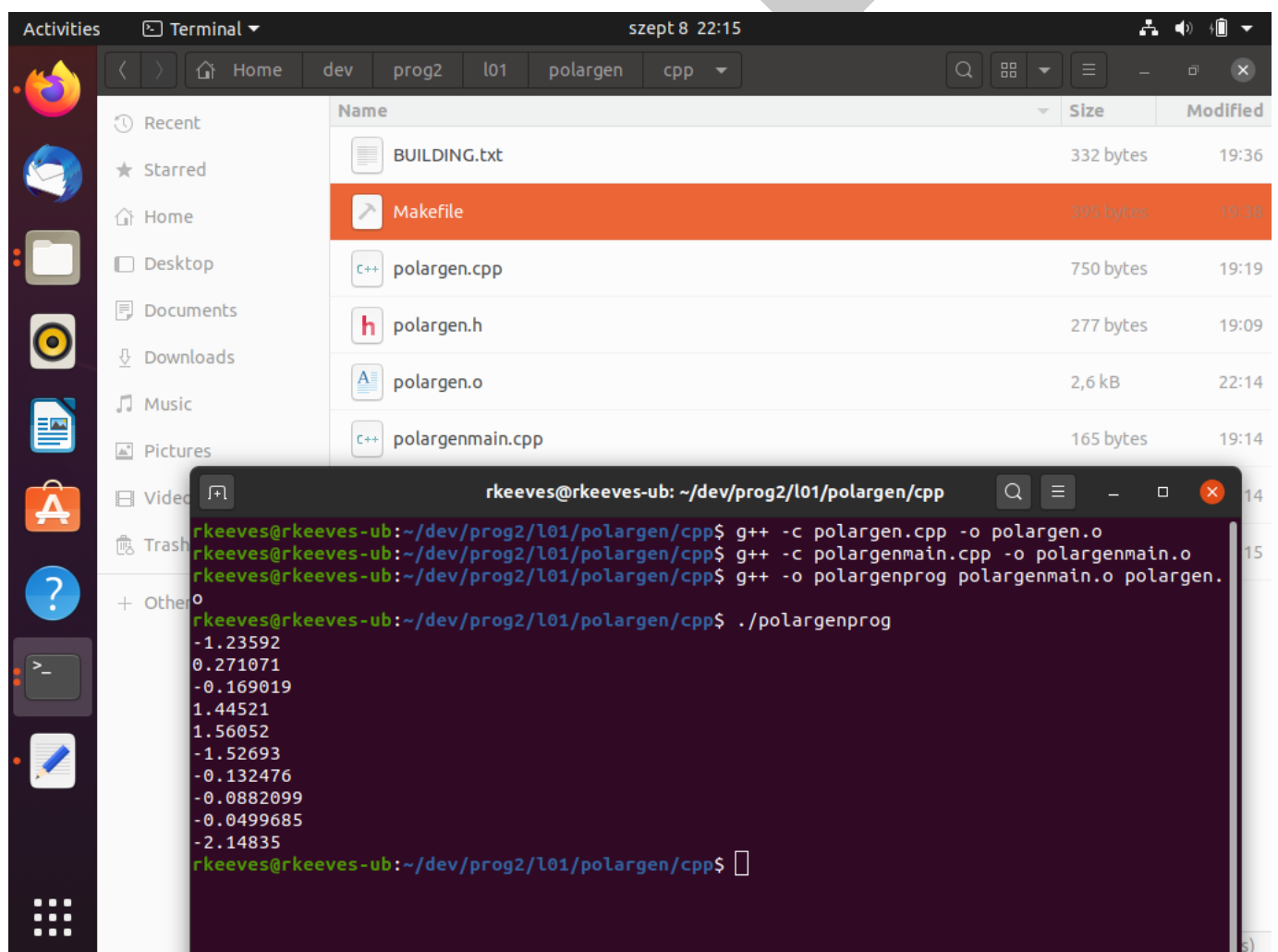
```
1 #include "polargen.h"  
2  
3 #include <cstdlib>  
4 #include <cmath>  
5 #include <ctime>  
6  
7 namespace prog2{  
8  
9     PolarGen::PolarGen() : store_empty(true), stored(0.0)  
10     {  
11         std::srand( std::time(NULL) );  
12     }  
13  
14  
15     double PolarGen::next()  
16     {  
17         if(store_empty){  
18             double u1,u2,v1,v2,w;  
19             do{  
20                 u1 = std::rand() / (RAND_MAX + 1.0);  
21                 u2 = std::rand() / (RAND_MAX + 1.0);  
22                 v1 = 2 * u1 - 1;  
23                 v2 = 2 * u2 - 1;  
24                 w = v1 * v1 + v2 * v2;  
25             }while(w > 1);  
26             double r = std::sqrt((-2 * std::log(w)) / w);  
27             stored = r * v2;  
28             store_empty = !store_empty;  
29             return r * v1;  
30         }else{  
31             store_empty = !store_empty;  
32             return stored;  
33         }  
34     }  
35 }
```

```
34     }  
35 } /* prog2 */  
36
```

Egyébként jó szokás saját namespace-ben dolgozni, emiatt vezettem be ezen fájlokba a prog2 namespace-t. Ha mindez meg volt akkor már csak a "main" van hátra.

```
1 #include <iostream>  
2 #include "polargen.h"  
3  
4 int main(int argc, char** argv){  
5     prog2::PolarGen g;  
6     for(int i = 0; i < 10; ++i)  
7         std::cout<<g.next()<<std::endl;  
8 }
```

A fordítás során ugye nem fordíthatjuk először a main-t. Előtte szükségünk van arra az objektumra, mely a polargen fordításából jön. Alább látható ez a folyamat.



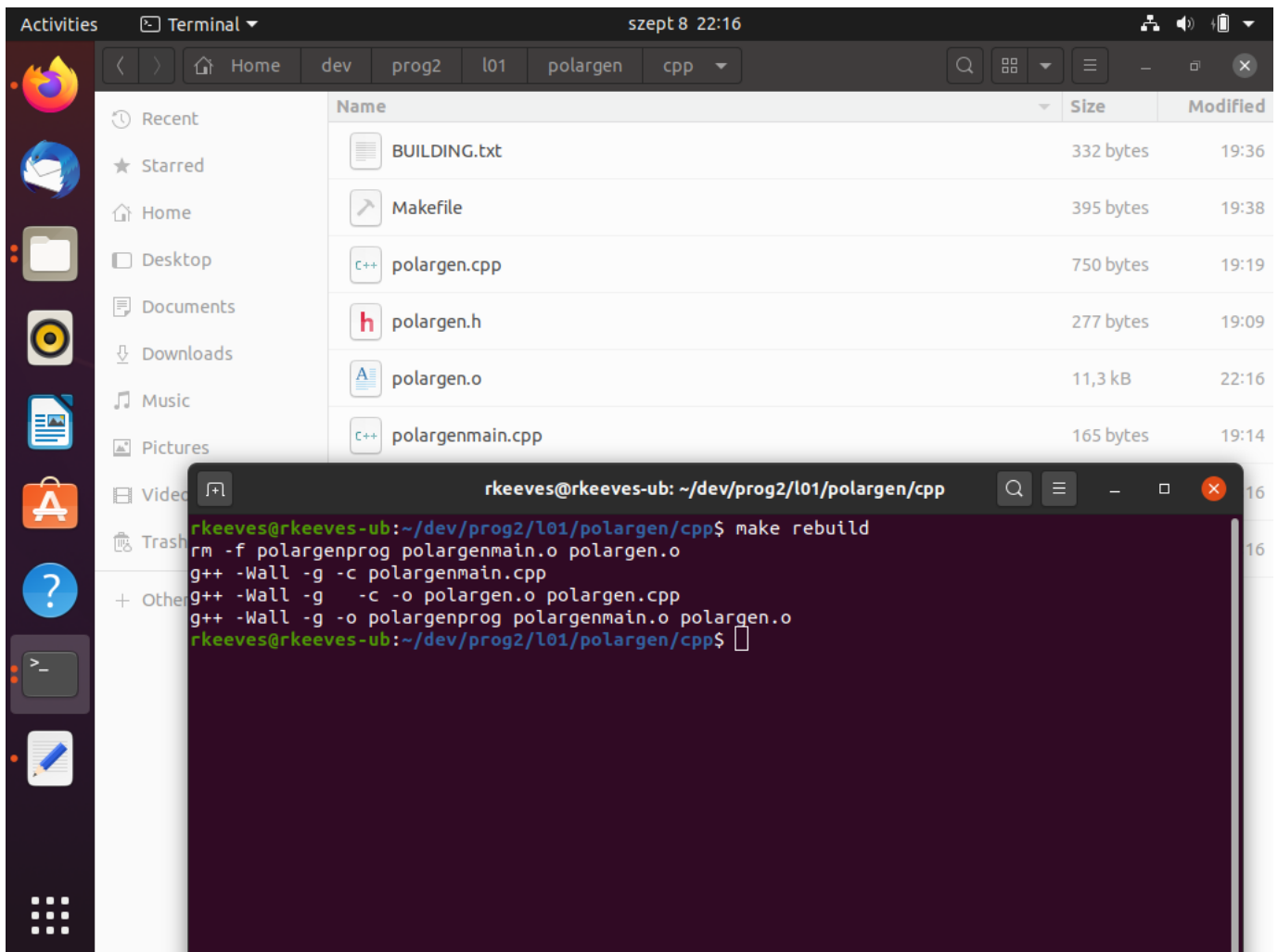
3.2. ábra. Manual Build



Persze egy olyan projektnél ahol 50+ file-unk van, ott érdemes lehet ezt automatizálni. Erre szolgálnak a makefile-ok. Alább egy primitív egyszerűségű példa.

```
1 # This is a really simplistic makefile
2 # For real projects use CMake, or Premake5
3
4 CXX = g++
5 CXXFLAGS = -Wall -g
6
7 rebuild: clean build
8
9 build: polargenmain.o polargen.o
10     $(CXX) $(CXXFLAGS) -o polargenprog polargenmain.o polargen.o
11
12 polargenmain.o: polargenmain.cpp polargen.h
13     $(CXX) $(CXXFLAGS) -c polargenmain.cpp
14
15 polargen.o: polargen.h
16
17 clean:
18     rm -f polargenprog polargenmain.o polargen.o
```

Alább egy példa a build-re make-el.



3.3. ábra. Make Build

A make feletti szintet a CMake képviseli, ezt nagyon széles körben használják és pont olyan borzalmas is mint minden amit sok ember szeret. Másik megoldás a premake ami nagyon jó annak aki szereti az átlátható dolgokat, a Lua-t. Apró hátulütője, hogy jó esetben kinevetik miatta az embert rossz esetben pedig ki is közösítik.

## 3.2. Homokozó

Írjuk át az első védelési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani 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!

Long story short, itt a Java verzió. Egyszerűen átemeltük c++-ból emiatt nagyon csúnya. A Java és a C++ nagyon komolyan eltér egymástól ennek hatásai meg is látszódnak.

C++ esetben ha olyan típust használunk mely esetén nem értelmezett a less than, és az equality operátorok akkor fordítási hiba lesz. Java esetében ezt például a Comparable interface öröklésével kényszeríthetjük ki, mint pl.

Igazi nagy különbség hogy míg C++ esetében a rule of five miatt copy, move ctor/assignment-re szükség volt, addig Java esetén elég egy deep copy.

Ami számomra problémát okozott, az az, hogy a T paraméternek Cloneable-nek kellene lennie. Eddig ok is, de nem hívhatjuk a java.lang.Object.clone-t, mert az alapvetően protected visibility-vel rendelkezik. So good luck using that out of the box :D (I'd personally consider accepting a functor which is responsible for the copying of any instance of T, but that might be wrong.)

```
1 import java.util.Arrays;
2 import java.util.function.BiConsumer;
3
4 /**
5  * Looks ugly af, but I wanted to mimic our c++ solution as closely as ↵
6  * possible.
7  * This is terrible in a lot of sense.
8  * For me, the fact that BinTree requires T to be Comparable is a crime ↵
9  * against humanity,
10 * but the task wass to directly copy the c++ src, so here we go folks!
11 */
12 public class BaseTree<T> {
13     class Node{
14
15         Node(T val) {
16             super();
17             this.val = val;
18             this.count = 1;
19             this.left = null;
20             this.right = null;
21         }
22
23         public Node(T val, BaseTree<T>.Node left, BaseTree<T>.Node right) {
24             super();
25             this.val = val;
26             this.count = 1;
27             this.left = left;
28             this.right = right;
29         }
30
31         Node copy(){
32             return new Node(this.val,
33                 (this.left == null ? null : this.left.copy()),
34                 (this.right == null ? null : this.right.copy()));
35         }
36
37         T val;
```

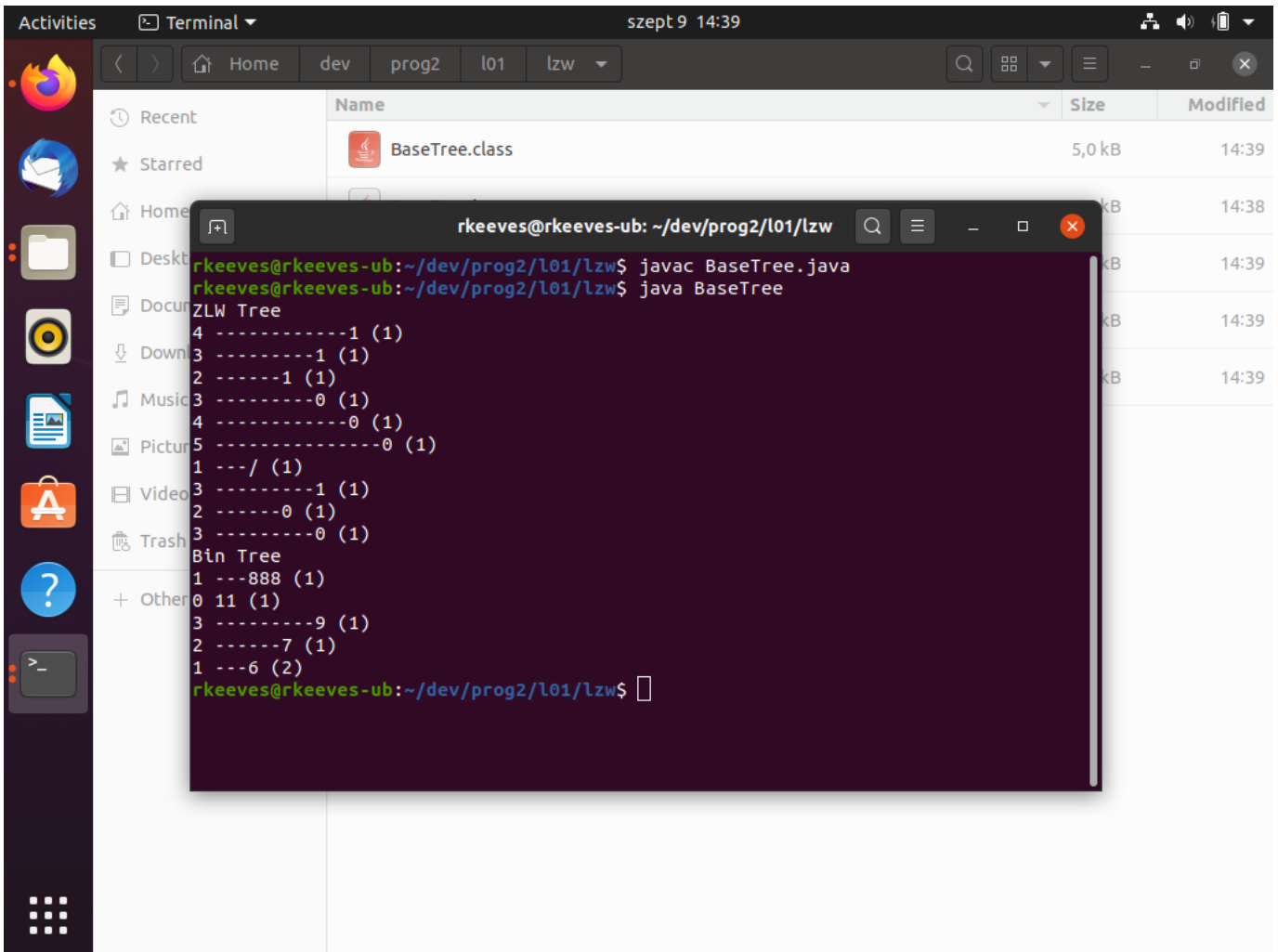
```
38     int count;
39
40     Node left;
41
42     Node right;
43 }
44
45 BaseTree() {
46     this.root = null;
47     this.treep = null;
48 }
49
50 BaseTree(T val) {
51     this.root = new Node(val);
52     this.treep = root;
53 }
54
55 BaseTree(Node root, Node treep) {
56     this.root = root;
57     this.treep = treep;
58 }
59
60 public BaseTree<T> copy() {
61     if(this.root == null)
62         return new BaseTree<T>(null, null);
63     else
64         return new BaseTree<T>(root.copy(), null);
65 }
66
67
68
69 Node cp(Node node, Node treep)
70 {
71     Node newNode = null;
72
73     if(node != null)
74     {
75         newNode = new Node(node.val);
76
77         newNode.left = cp(node.left, treep);
78         newNode.right = cp(node.right, treep);
79         newNode.count = node.count;
80
81         if(node == treep)
82             this.treep = newNode;
83     }
84
85     return newNode;
86 }
87
```

```
88 public void traverse(BiConsumer<Integer,Node> cons) {
89     traverse(0, this.root, cons);
90 }
91
92 private void traverse(int depth, Node n, BiConsumer<Integer,Node> ←
93     user_fun) {
94     if(n != null){
95         traverse(depth+1, n.right, user_fun);
96         user_fun.accept(depth,n);
97         traverse(depth+1, n.left, user_fun);
98     }
99 }
100
101 public void pretty_print_tree() {
102     traverse(1, this.root, this::pretty_print_node);
103 }
104
105 public void pretty_print_node(Integer depth, Node n) {
106     StringBuilder sb = new StringBuilder();
107     sb.append(depth)
108     .append(" ");
109     for(int i = 0; i < depth; ++i)
110         sb.append("---");
111     sb.append(n.val.toString())
112     .append(" (")
113     .append(n.count)
114     .append(") ");
115     System.out.println(sb.toString());
116 }
117
118
119 protected Node root;
120
121 protected Node treep;
122
123
124 public static class ZLWTree<T> extends BaseTree<T>{
125
126     public ZLWTree(T val_root, T val_left) {
127         super(val_root);
128         this.val_left=val_left;
129     }
130
131     public ZLWTree<T> add(T value)
132     {
133         if(value == null)
134             return this;
135         if(value.equals(val_left)) {
136             if(treep.left == null) {
```

```
137         treep.left = new Node(value);
138         treep = root;
139     } else {
140         treep = treep.left;
141     }
142 } else {
143     if(treep.right == null) {
144         treep.right = new Node(value);
145         treep = root;
146     } else {
147         treep = treep.right;
148     }
149 }
150 }
151 return this;
152 }
153
154 private T val_left;
155 }
156
157 public static class BinTree<T extends Comparable<T>> extends BaseTree<T>{
158
159     public BinTree() {
160         super();
161     }
162
163
164     public BinTree<T> add(T value)
165     {
166         if(value == null)
167             return this;
168
169         if(treep == null) {
170             root = treep = new Node(value);
171         } else {
172             int cmp = treep.val.compareTo(value);
173             if(cmp == 0) {
174                 treep.count++;
175             }else if(cmp > 0) {
176                 if(treep.left == null) {
177                     treep.left = new Node(value);
178                 } else {
179                     treep = treep.left;
180                     this.add(value);
181                 }
182             }else if(cmp < 0) {
183                 if(treep.right == null) {
184                     treep.right = new Node(value);
185                 } else {
186                     treep = treep.right;
```

```
187         this.add(value);
188     }
189 }
190 }
191 treep = root;
192
193     return this;
194 }
195
196 }
197
198 public static void main(String[] args) {
199
200     System.out.println("ZLW Tree");
201     ZLWTree<Character> zlw = new ZLWTree<>('/', '0');
202     "01111001001001000111"
203         .chars()
204         .mapToObj((i) -> (char) i)
205         .forEach(zlw::add);
206     zlw.pretty_print_tree();
207
208     System.out.println("Bin Tree");
209     BinTree<Integer> bt = new BinTree<>();
210     Arrays.asList(11, 6, 7, 888, 9, 6).forEach((c) -> bt.add(c));
211     bt.traverse(bt::pretty_print_node);
212 }
213 }
```

Compile + execution példa alább.



```
rkeeves@rkeeves-ub: ~/dev/prog2/l01/lzw
rkeeves@rkeeves-ub:~/dev/prog2/l01/lzw$ javac BaseTree.java
rkeeves@rkeeves-ub:~/dev/prog2/l01/lzw$ java BaseTree
ZLW Tree
4 -----1 (1)
3 -----1 (1)
2 -----1 (1)
3 -----0 (1)
4 -----0 (1)
5 -----0 (1)
1 ---/ (1)
3 -----1 (1)
2 -----0 (1)
3 -----0 (1)
Bin Tree
1 ---888 (1)
0 11 (1)
3 -----9 (1)
2 -----7 (1)
1 ---6 (2)
rkeeves@rkeeves-ub:~/dev/prog2/l01/lzw$
```

3.4. ábra. Lzw Java

### 3.3. Gagy

Az ismert formális „while ( $x \leq t \ \&\& \ x \geq t \ \&\& \ t \neq x$ );” 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! Apéldát építsd a JDK Integer.java forrására<sup>4</sup>, hogy a 128-nál inkluzív objektum példányokat poolozza!

Tanulságok, tapasztalatok, magyarázat...

### 3.4. Yoda

Írjunk olyan Java programot, ami java.lang.NullPointerException-el leáll, ha nem követjük a [Yoda conditions](#)-t!

Tanulságok, tapasztalatok, magyarázat...



### 3.5. Kódolás from scratch

Induljunk ki [ebből](#) a tudományos közleményből: és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! [Ha megakadsz, de csak végső esetben](#): (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Tanulságok, tapasztalatok, magyarázat...

### 3.6. EPAM: Java Object metódusok

Mutasd be a Java Object metódusait és mutass rá mely metódusokat érdemes egy saját osztályunkban felül-definiálni és miért. (Lásd még Object class forráskódja)

Tanulságok, tapasztalatok, magyarázat...

### 3.7. EPAM: Eljárásorientált vs Objektumorientált

Írj egy 1 oldalas értekező esszé szöveget, amiben összehasonlítod az eljárásorientált és az objektumorientált paradigmát, igyekezve kiemelni az objektumorientált paradigma előnyeit!

Tanulságok, tapasztalatok, magyarázat...

### 3.8. EPAM: Objektum példányosítás programozási mintákkal

Hozz példát mindegyik “creational design pattern”-re és mutasd be mikor érdemes használni őket!

Tanulságok, tapasztalatok, magyarázat...

## **III. rész**

### **Irodalomjegyzék**

DRAFT

### 3.9. Általános

- [CPP] Benedek, Zoltán Ács Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Szak Kiadó Kft. , 2013.
- [JAVA] Nyékyné Dr. Gaizler Judit et al., Judit, *Java 2 útikalauz programozóknak 5.0 I.-II.*, ELTE TTK Hallgatói Alapítvány , 2008.
- [PYTHON] Forstner, Bertalan, Ekler, Péter, Ács Kelényi, Imre, *Bevezetés a mobil programozásba. Gyors prototípus-fejlesztés Python és Java nyelven*, Szak Kiadó Kft. , 2008.

### 3.10. My Little Ponys

- [CHISOMORPH] Sorensen, Morten Heine B, *Lectures on the Curry-Howard Isomorphism*, [Pdf](#) , 1998.
- [DENOTATIONALSEMANTICS] Allison, Lloyd, *A practical introduction to denotational semantics*, [Cambridge University Press](#) , 1986.
- [NANDTOTETRIS] Nisan, Noam, *The Elements of Computing Systems*, [Homepage of the book](#) , 2005.
- [PROGLANGS] Harper, Robert, *Practical Foundations for Programming Language*, [Carnegie Mellon University](#) , 2016.

Köszönet illeti Bátfai Norbert Tanár Urat, továbbá a tárgy gyakorlat tartóit, és az [UDPROG](#)-ot .