Jegyzőkönyv

Programozás 2 tárgyra

2018/2019-es tanév első félév

Készítette: Oláh Adél

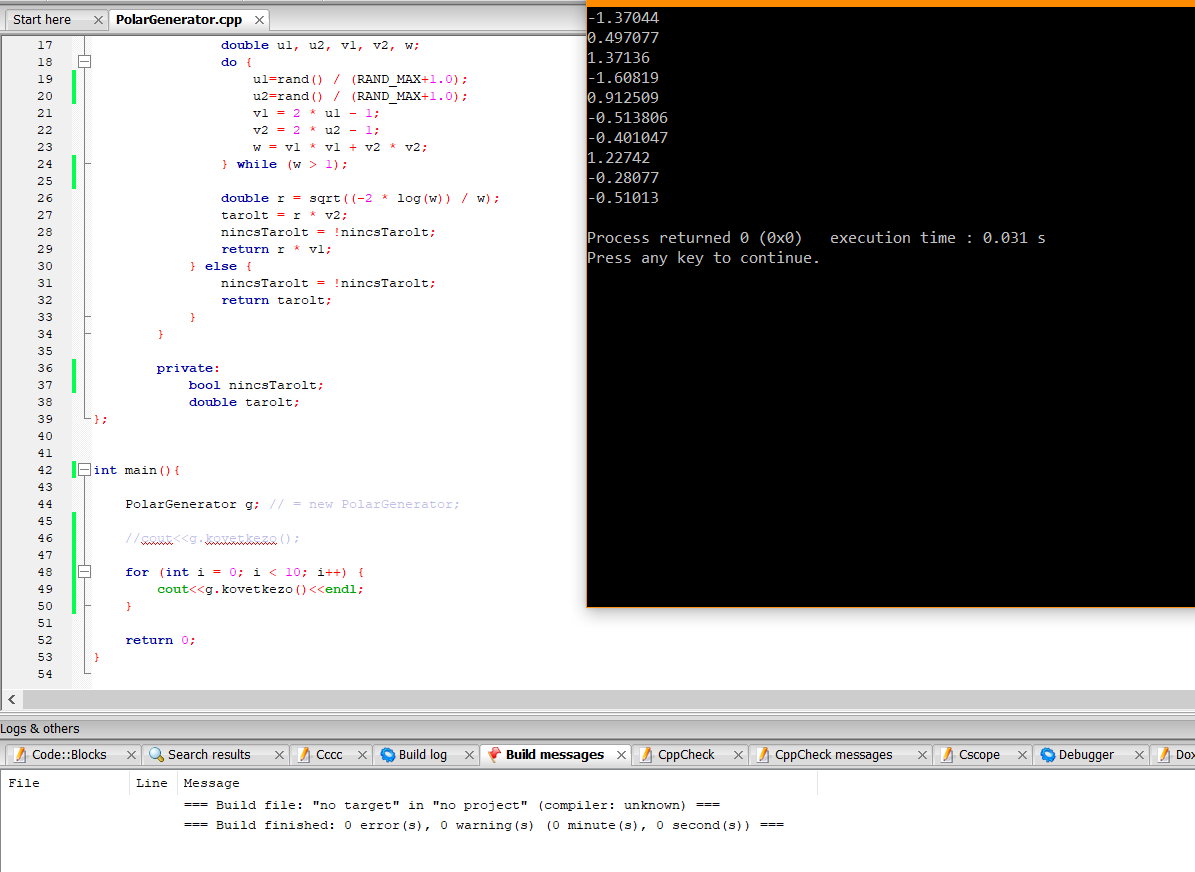
Forráskódok: *https://github.com/olahadel/UDPROG/tree/master/prog2*

Első hét:

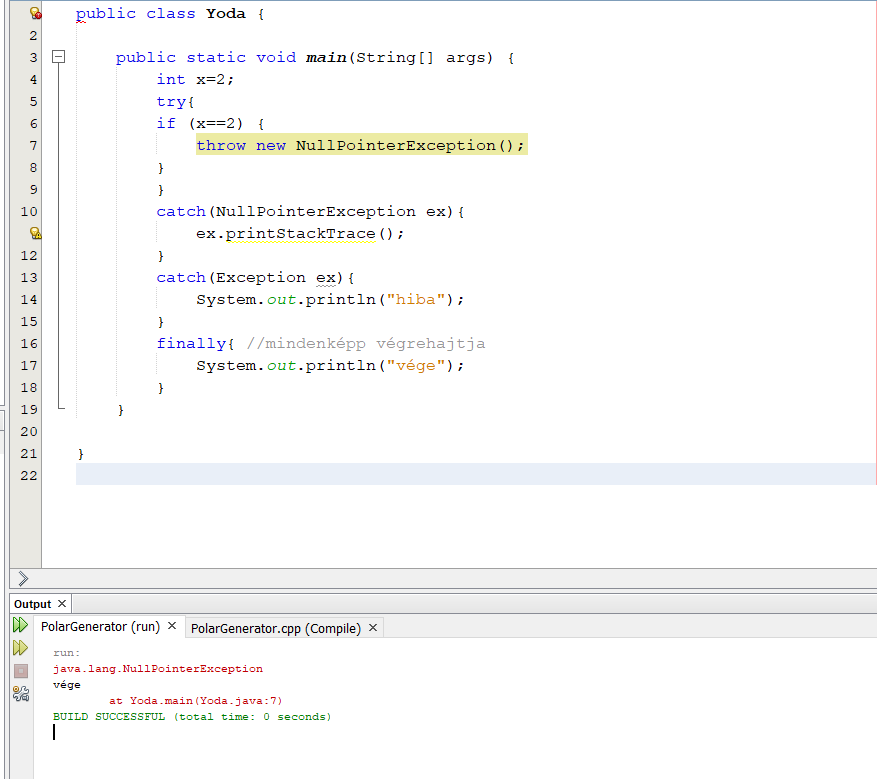
**OO szemlélet** (4. héten befejezve)  
**Homokozó** (elkezdve)  
**„Gagyi”** (elkezdve)  
**Yoda** (4. héten befejezve)  
**Kódolás from scratch** (2. héten befejezve)

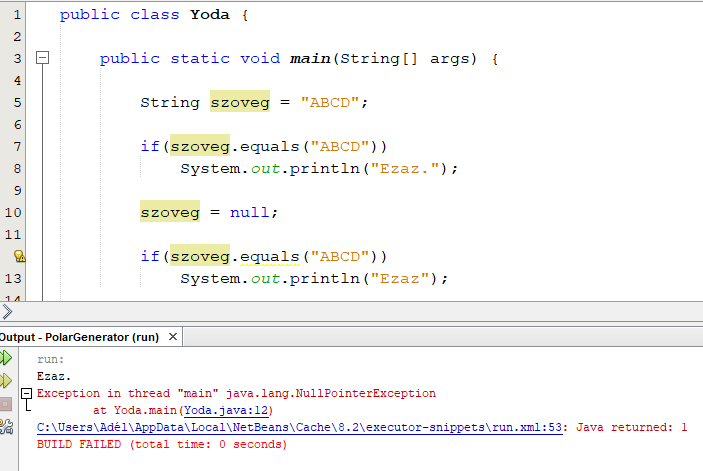
2018.09.12.: Az első feladaton (**PolarGenerator**) dolgoztam, a c++ fájl még nem fut le.

2018.10.07.: Befejeztem az első heti **OO szemlélet** feladatot. Az eredeti Java változathoz hasonlóan ez is visszaadja a várt 10 számot.  
A Tanár Úr által belinkelt diasorban az OpenJDK (illetve az Oracle JDK) általi OO szervezés is ugyanaz, mint ebben a megoldásban. A diasorban szereplő kódban (továbbá „eredeti”) láthatunk egy nextNextGaussian és egy haveNextGaussian nevezetű változót, amely ebben a megoldásban sorrendazonosan a tarolt, illetve a nincsTarolt nevet kapta. Ezeknek megfelelően az eredetiben a nextGaussian() függvény megfelel a kovetkezo() függvénynek.  
A függvény alkalmazása esetén minden második (páratlan számú) híváskor fut le az algoritmus és először az r\*v1, majd az r\*v2 értéket adja vissza.   
Az eredeti kódban is ezt láthatjuk, csak míg ebben a logikai változó igaz értékről indul, az eredetiben hamisról, és ennek megfelelően cserélődik fel az if-else rész is.

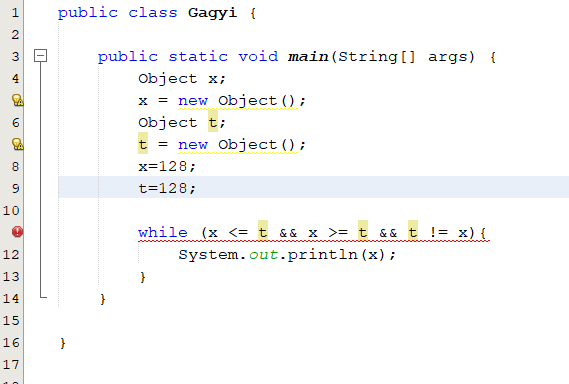


2018.09.14.: Kész a **Yoda** feladat, az én értelmezésemben.



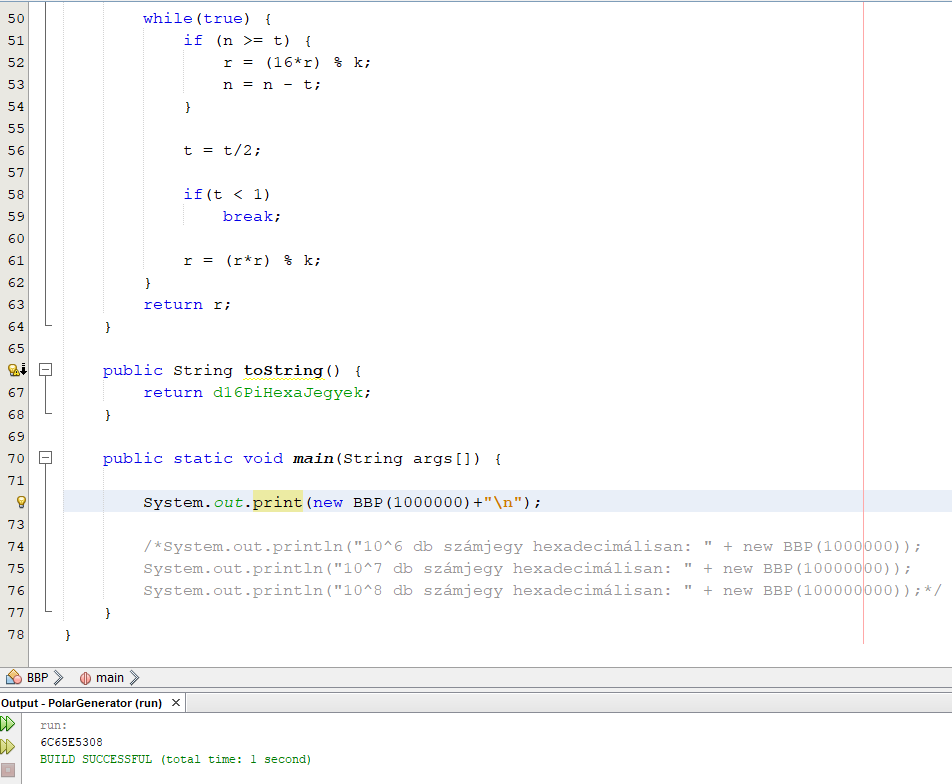
2018.10.07.: Az első heti **Yoda** feladatot átírtam, így most már kikényszerítés nélkül dob kivételt. A Yoda conditions azt jelenti az én értelmezésemben, hogy a feltételvizsgálatban a kifejezés két részét a szokásos, hagyományos sorrendhez képest felcseréljük. A feladat szövege alapján kivételt akkor kell kapnunk, ha a szokásos sorrendet használjuk, nem pedig a megfordított Yoda sorrendet.   
Ennek megfelelően az én megoldásomban is kivételt kapunk a Yoda sorrend esetén.  


2018.09.14.: Elkezdtem a **Gagyi** feladatot, de a megoldás még nincs meg. Nem sikerült értelmeznem a feladat szövegét.



2018.09.15.: Elkezdtem a **Homokozó** feladatot, eddig a kód átírását kezdtem el. A Java nyelvet még kevésbé ismerem a C++ - hoz képest, ezért még vannak benne problémák.

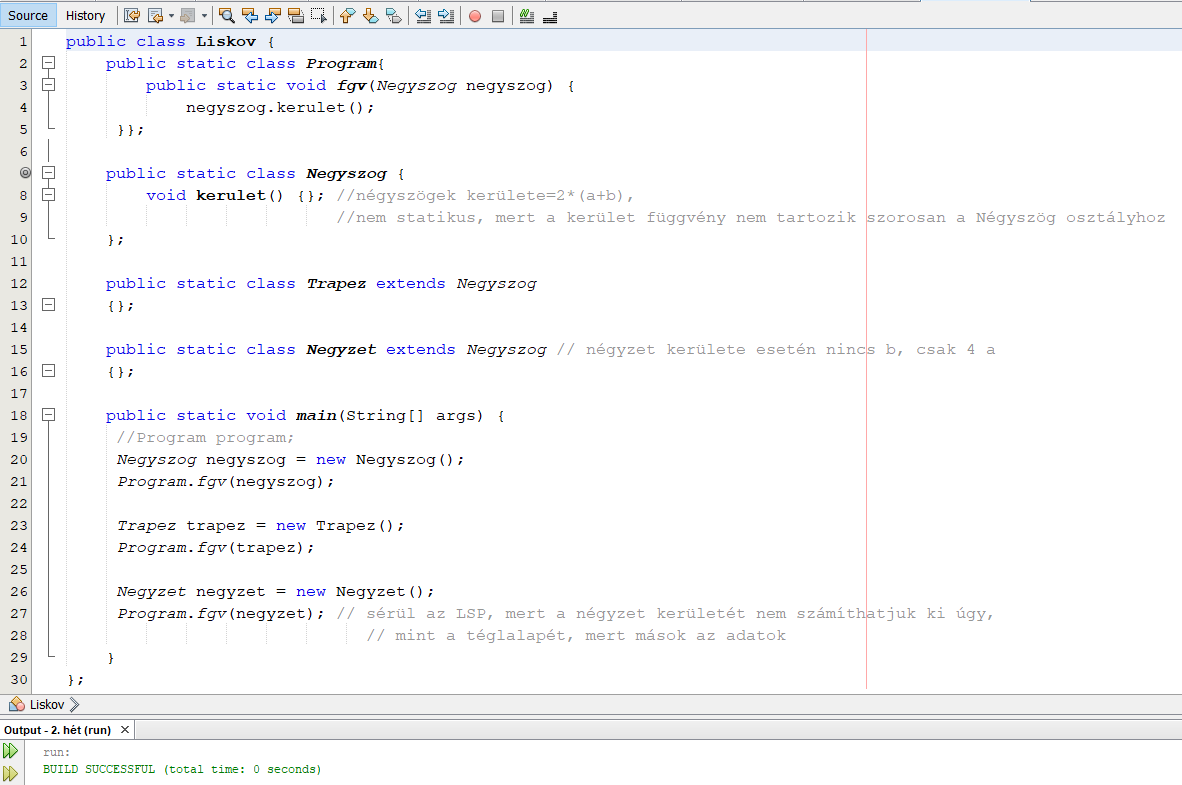
2018.09.16.: A **BBP algoritmust** tanulmányoztam a cikkből és elkezdtem dolgozni a Java kódon. (A második héten fejeztem be, mert akkor sikerült megérteni a forráskódot.)



2. hét:

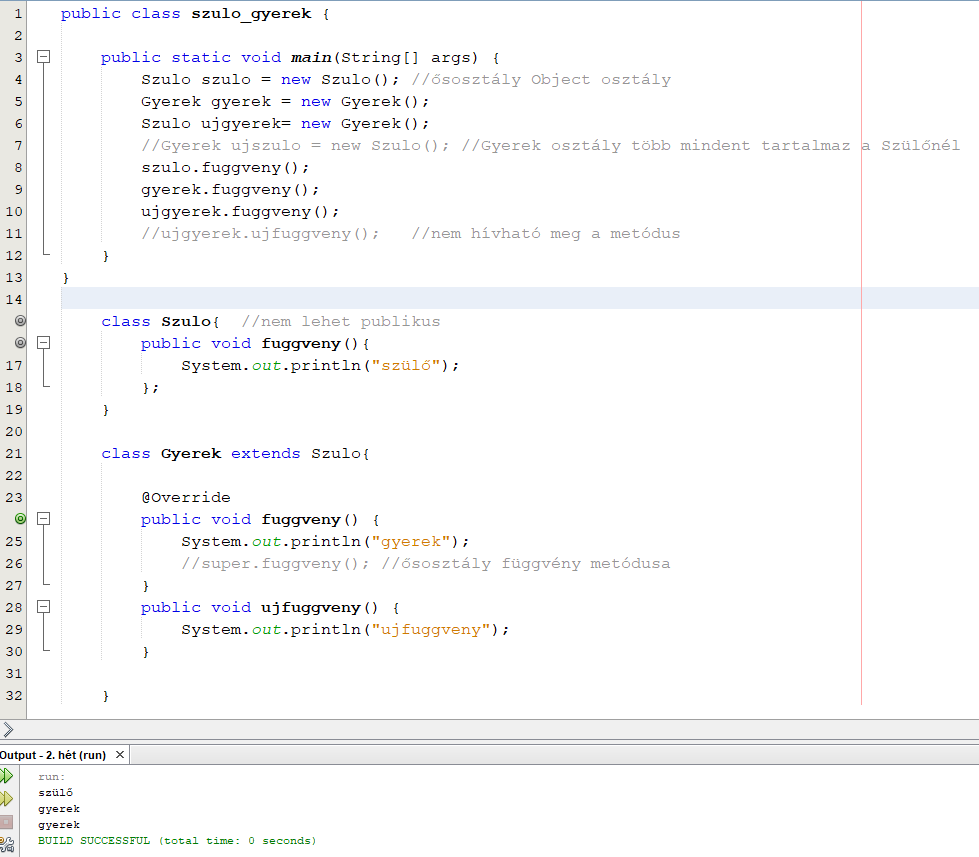
**Liskov helyettesítés sértése** (tárgyhéten befejezve) **Szülő-gyerek** (tárgyhéten befejezve) **Anti-OO** (tárgyhéten befejezve) **Hello, Android!** (elkezdve) **Ciklomatikus komplexitás**

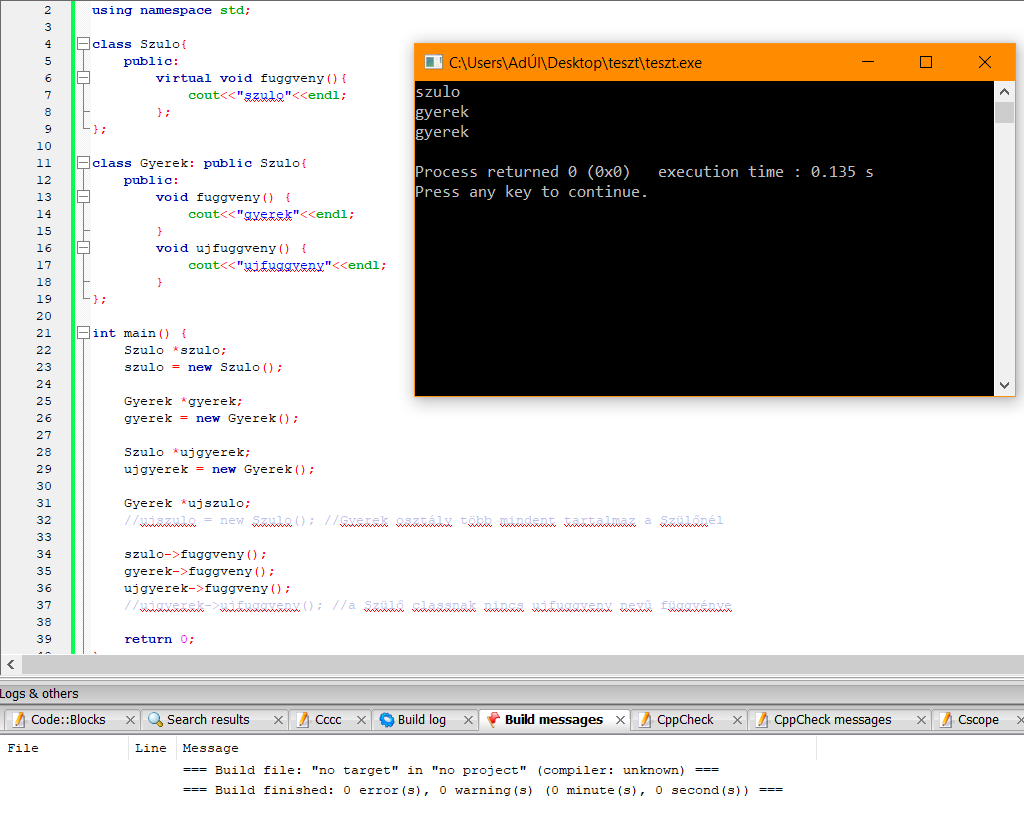
2018.09.19.: **Liskov elv megsértése** feladatra kész a C++, illetve Java kód. A feladatot a négyszögek kerületének kiszámításával szerettem volna bemutatni. Azért sérül az LSP a programban, mert a négyzet kerületét másképpen számítjuk ki, mint egy négyszögét; általános négyszög esetén A B C D bemenő paramétereink vannak, míg négyzet esetén csak A bemenő adatunk van. Általános trapéz esetén 4 különböző oldalunk van, ezért ott nem sérül az elv.



2018.09.22.: Kész a **Szülő-gyerek** feladat Java-ban. A kommentelt részekben látszik az, ami nem működik.   
A Szülő osztályt példányosíthatjuk a Gyerek osztály konstruktorával, de fordítva nem. A Gyerek osztály egy „bővebb” osztály lehet, mint a Szülő osztály, ezért az nem probléma, ha egy Szülő példánynak foglalunk le több helyet (amennyi egy Gyerek példánynak lenne szükséges). Ellenben a Szülő konstruktorával nem hozhatunk létre Gyerek példányt, mert ha a Gyerek osztály bővebb, akkor a Szülő konstruktora által lefoglalt hely nem feltétlenül biztosítaná a helyet a Gyerek elemei számára.  
Az ujgyerek.ujfuggveny() teljesíti igazából a feladatot. Az „ősön keresztül csak az ős üzenetei küldhetők” a feladat szerint. Az ős a Szülő osztály az üzenet pedig a fuggveny() metódus. Az ujgyerek.ujfuggveny() metódus esetén az ujgyerek a Szülő osztály egy példánya, míg az ujfuggveny a Gyerek osztály egy függvénye. Ez mutatja meg, hogy a szülő osztályon keresztül a gyerek osztály metódusa nem használható. (A Gyerek osztály példányán keresztül a Szülő metódusa használható bármilyen probléma nélkül, ahogyan ezt a program lefutása is mutatja.)

2018.09.24: Az **Android**os feladatot elkezdtem, az Android Stúdiót letöltöttem a feladatokkal együtt. A programot még nem sikerült megfelelően használni.



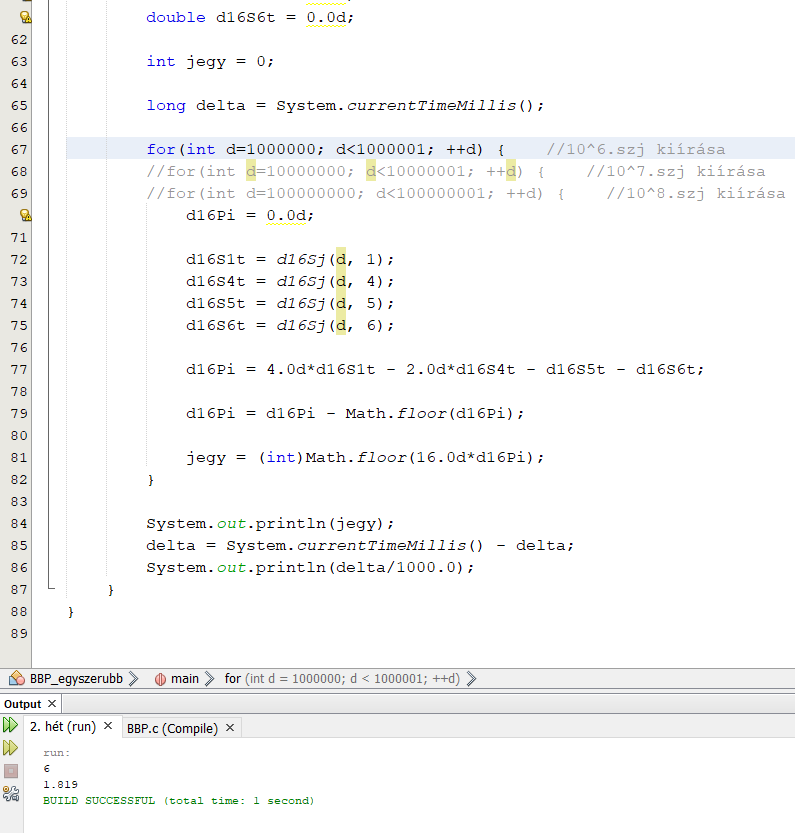
C++ -ban:

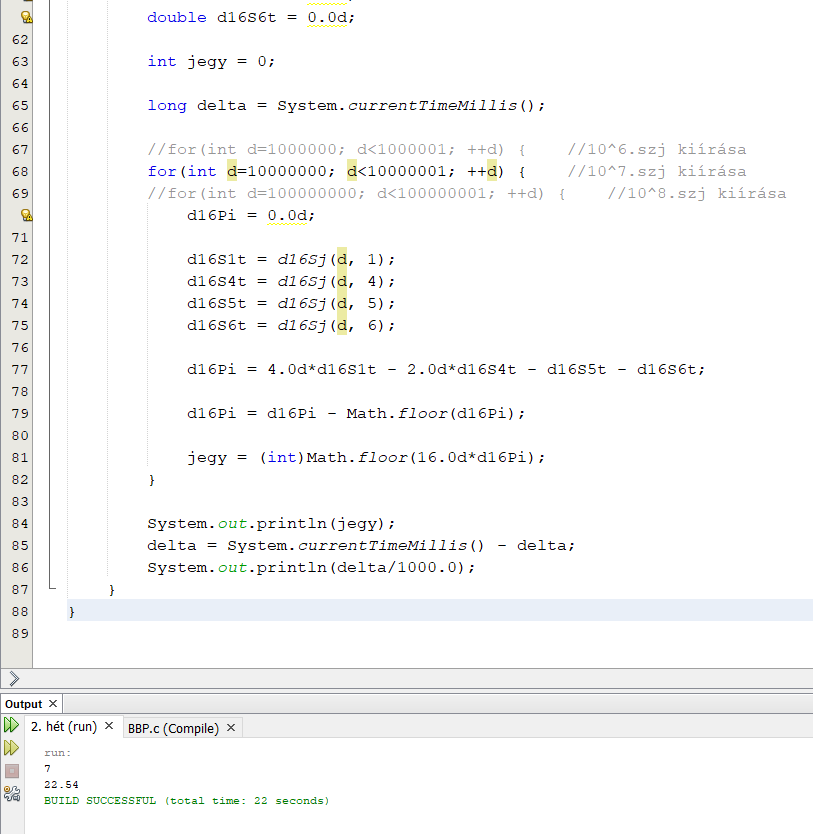
2018.09.25.: **Anti OO** feladat futási idők

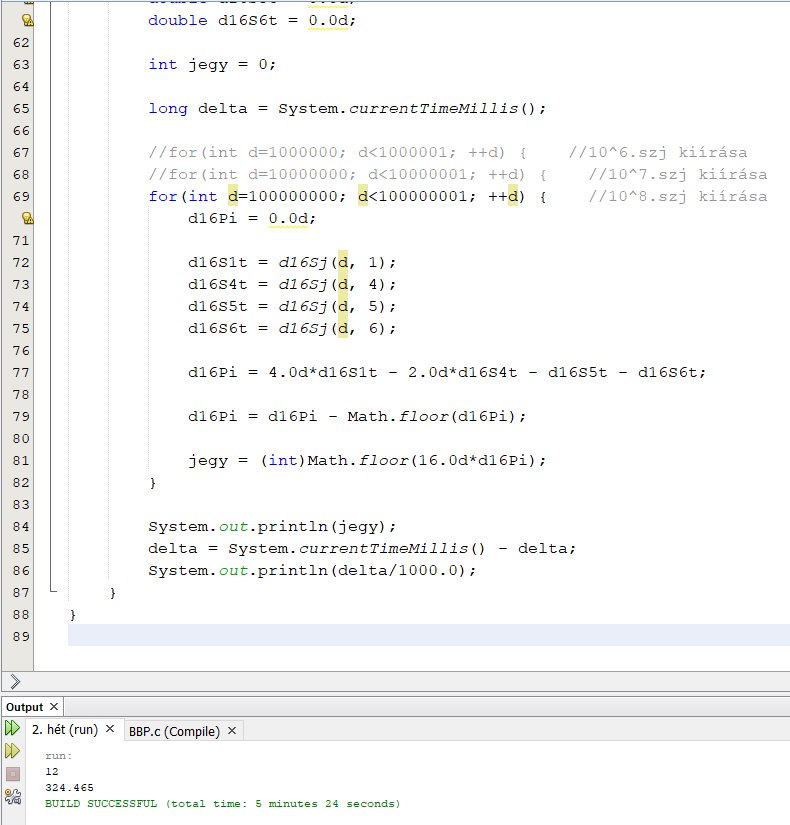
Összesítve:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10^6. jegy | 10^7. jegy | 10^8. jegy |
| Java | 1,819 s | 22.54 s | 324,465 s |
| C | 1,313 s | 15,734 s | 228,237 s |
| C# | 2,846 s | 32,347 s | 564,417 s |
| C++ | 1,31 s | 15,765 s | 225,103 s |

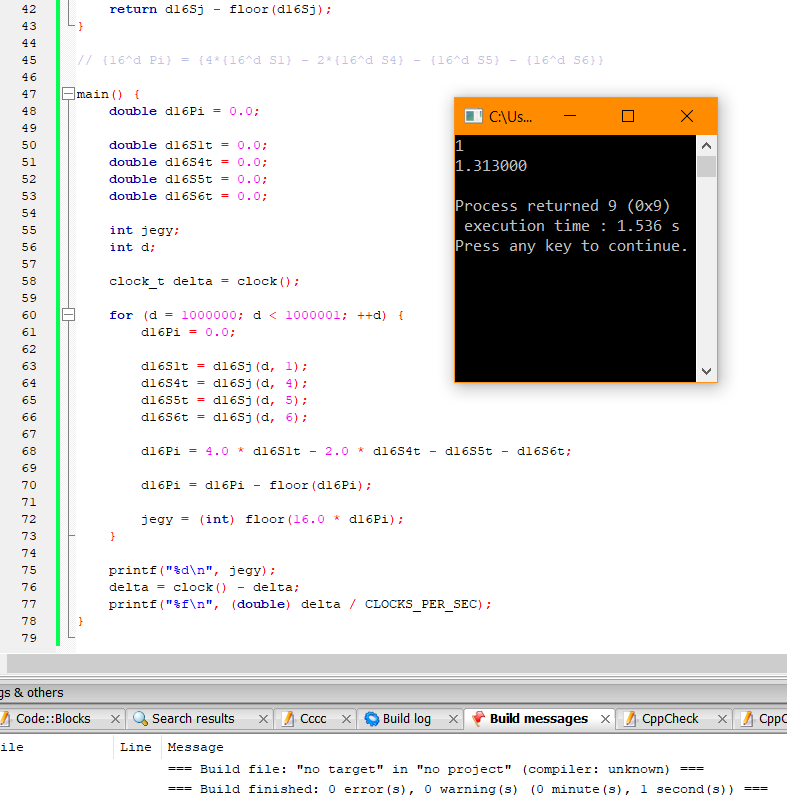
Javaban:

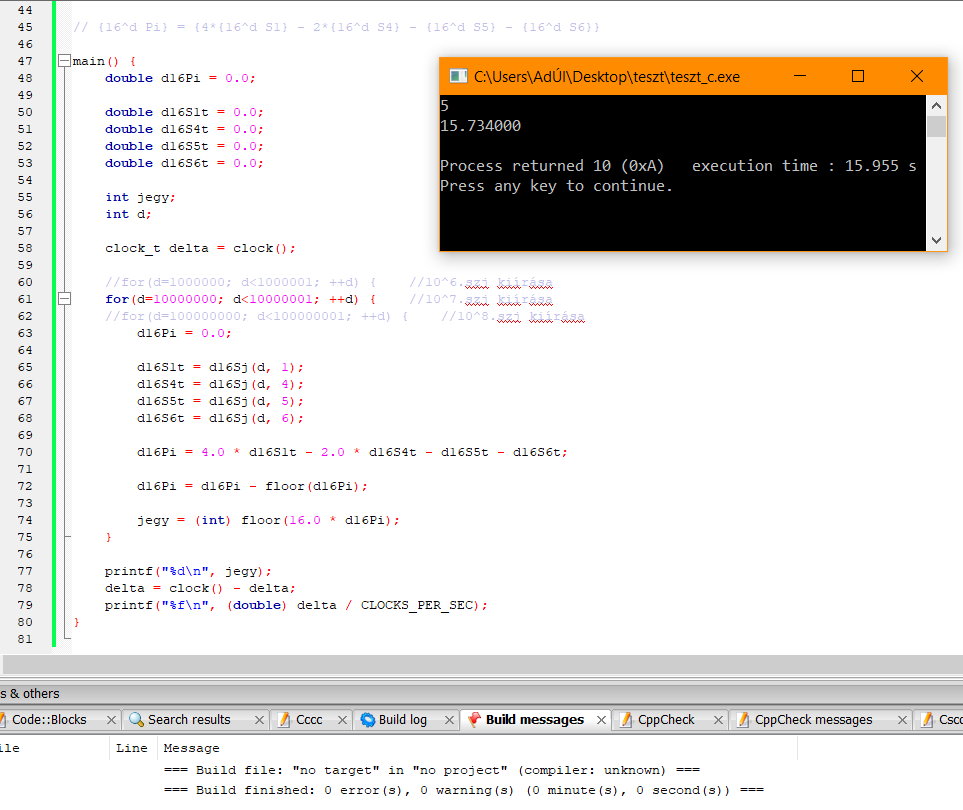
10^6. jegy:

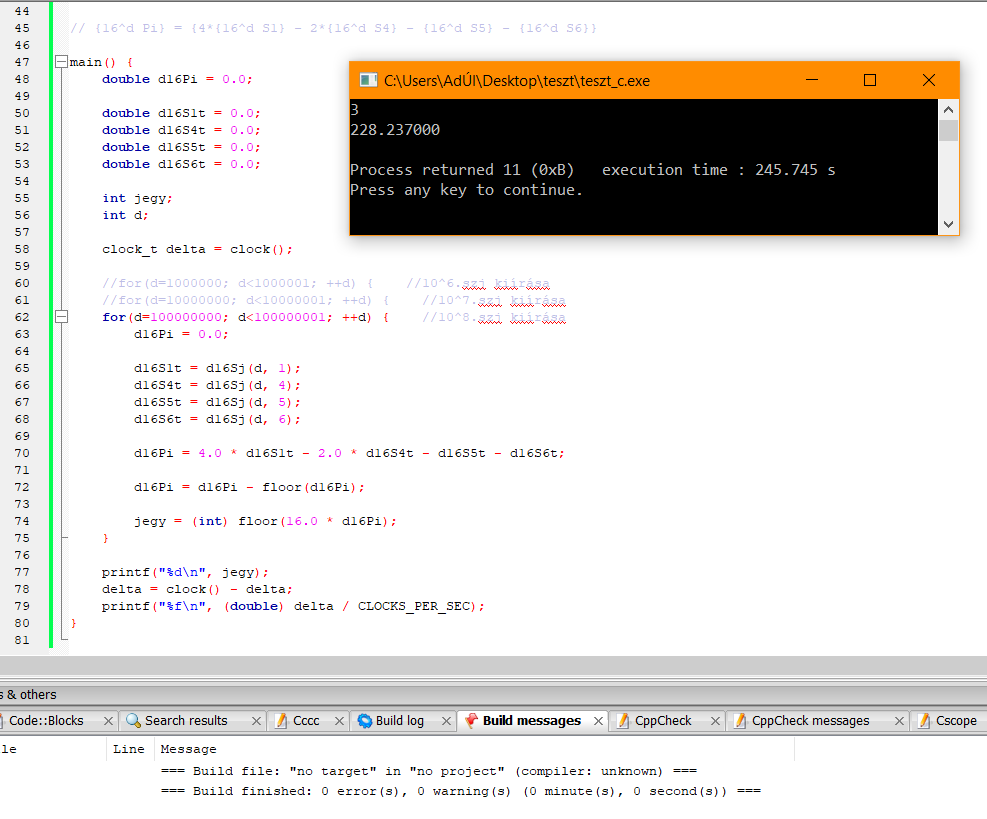
10^7.jegy: 

10^8. jegy:

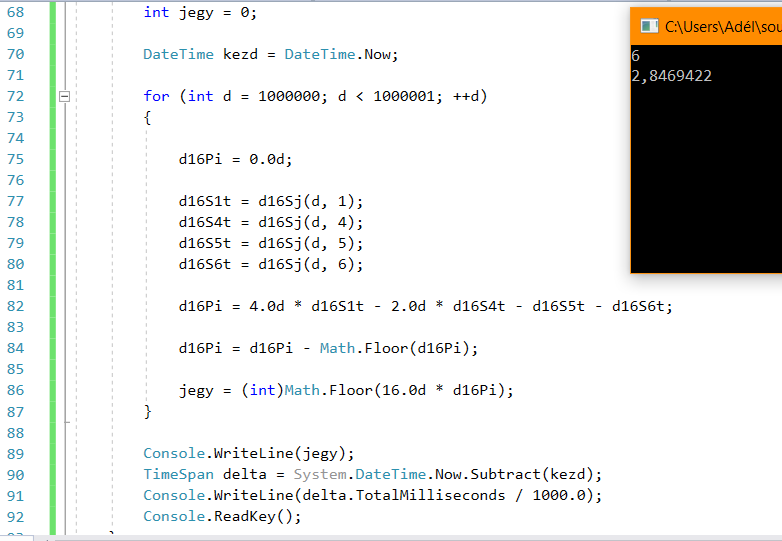
C-ben:

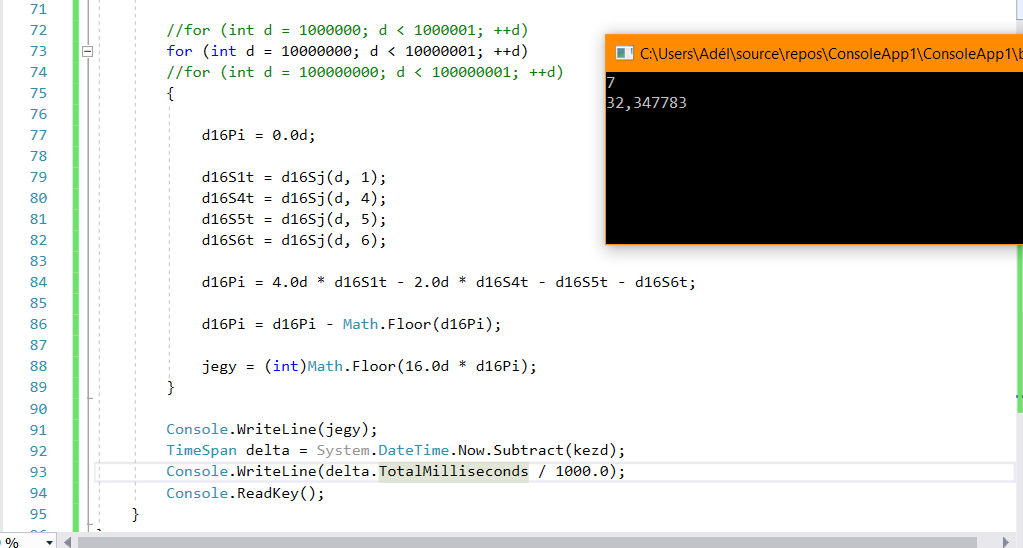
10^6. jegy:

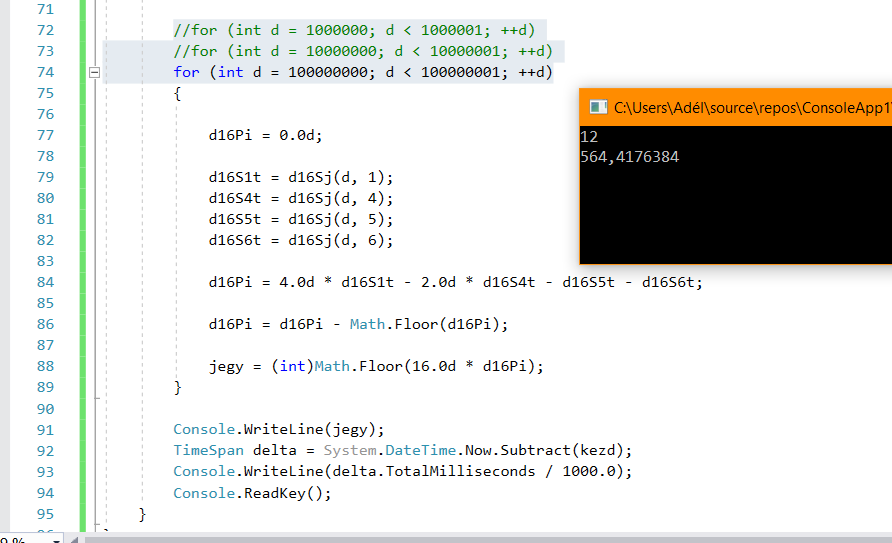
10^7. jegy: 

10^8. jegy: 

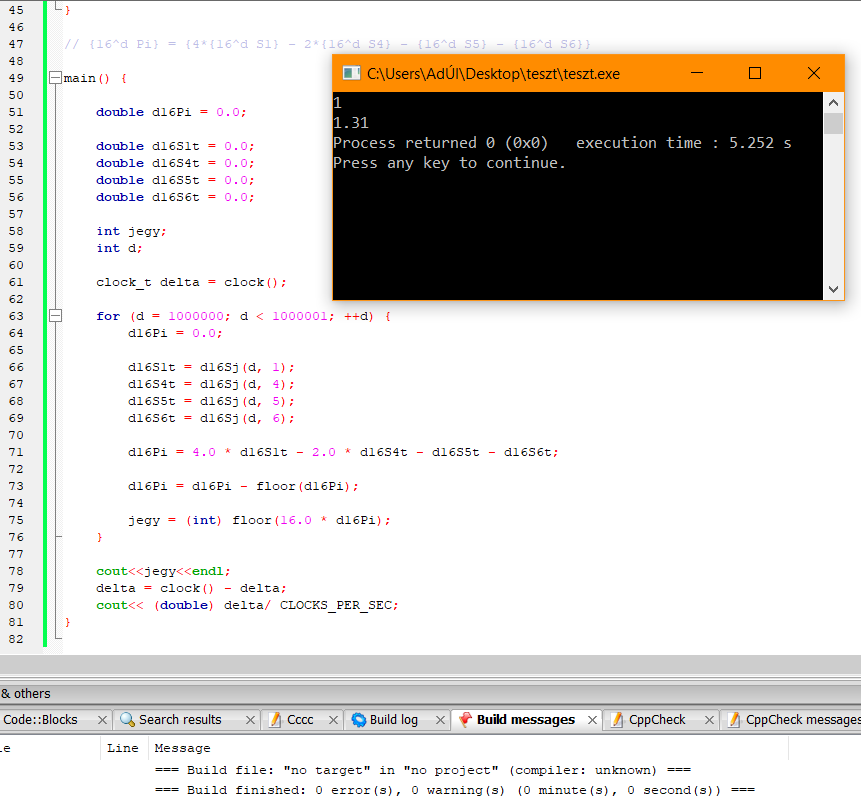
#C-ben:

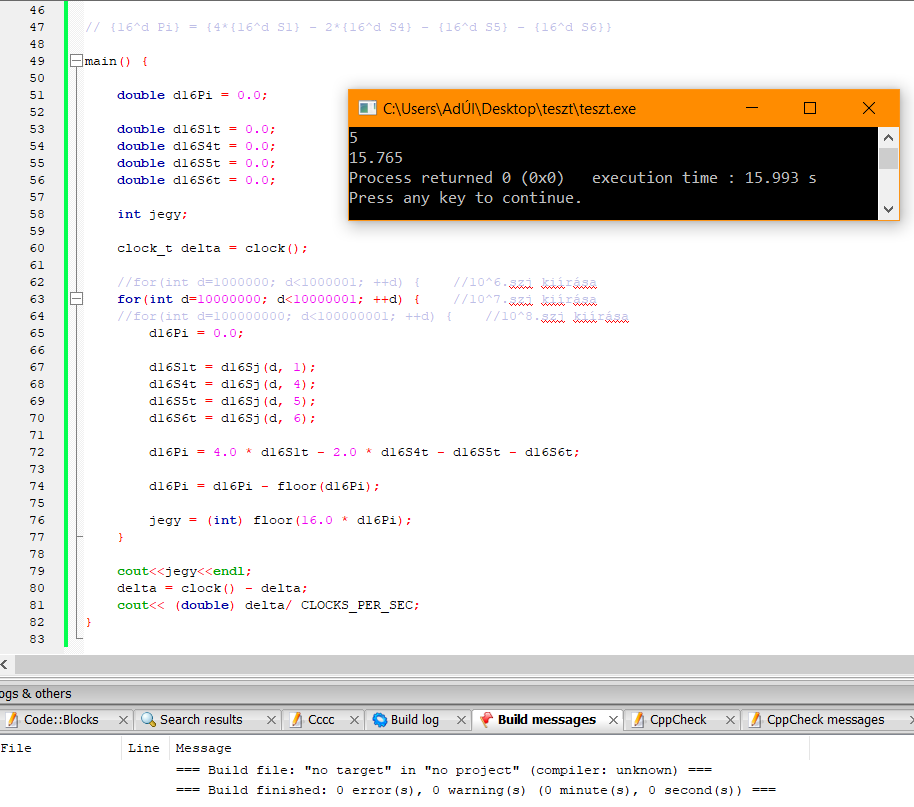
10^6. jegy:

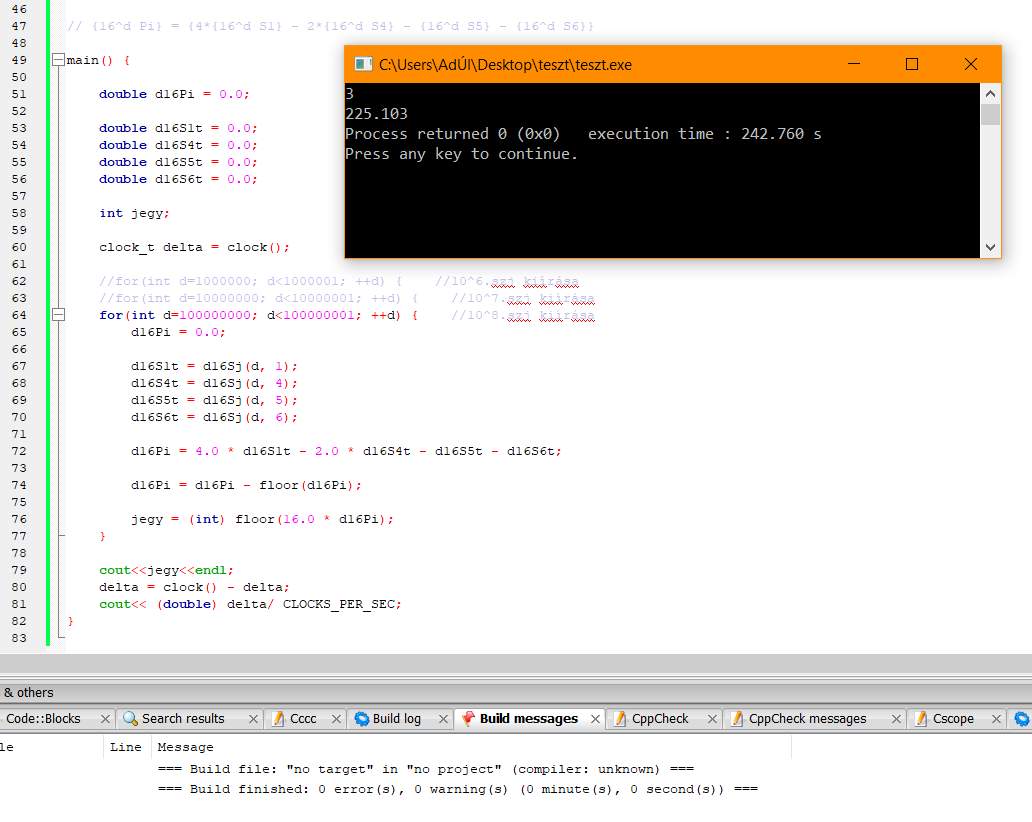
10^7. jegy: 

10^8. jegy: 

C++ -ban:

10^6. jegy:

10^7. jegy: 

10^8. jegy: 

3. hét

Reverse engineering UML osztálydiagram

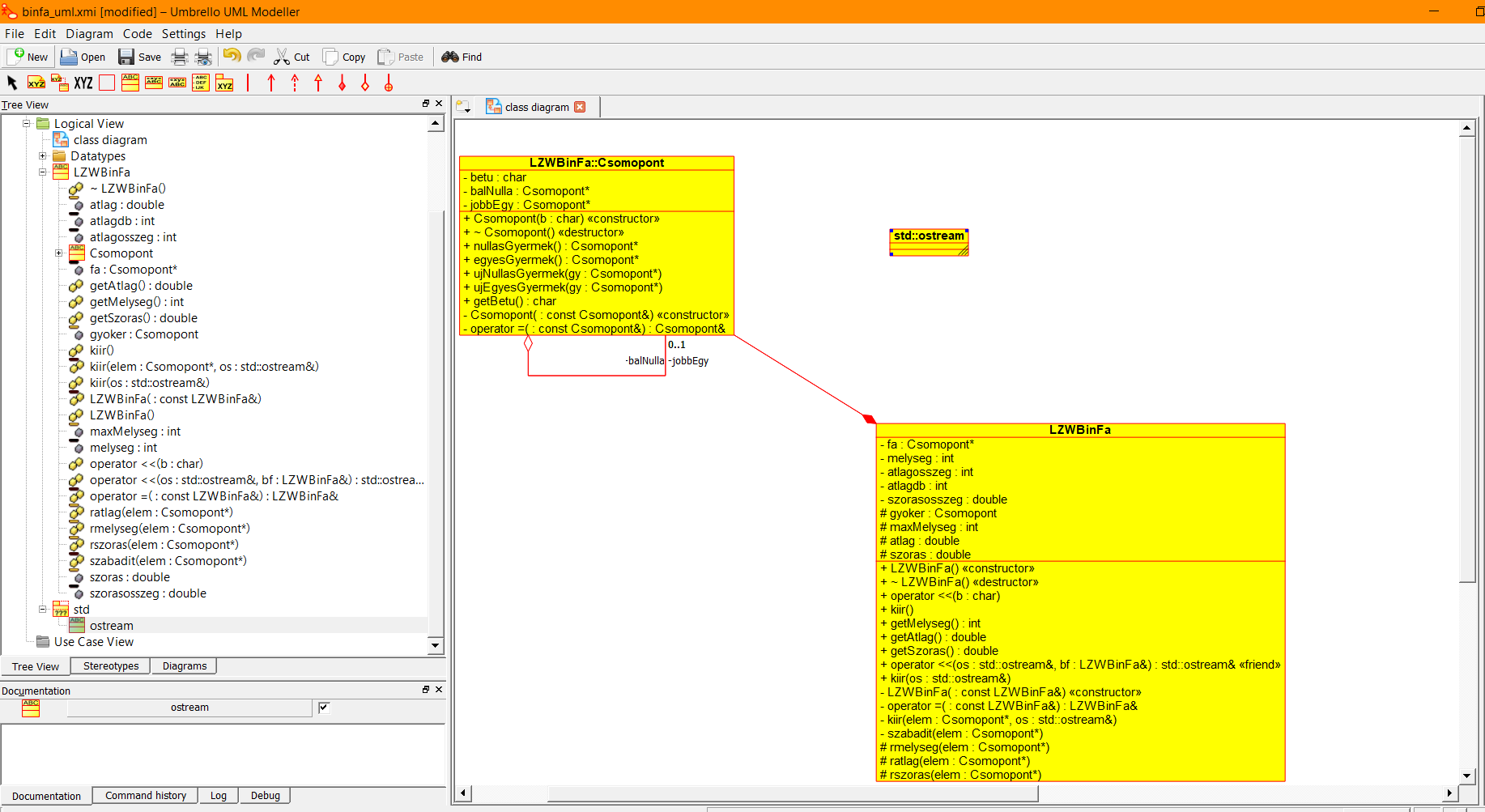
Forward engineering

Egy esettan

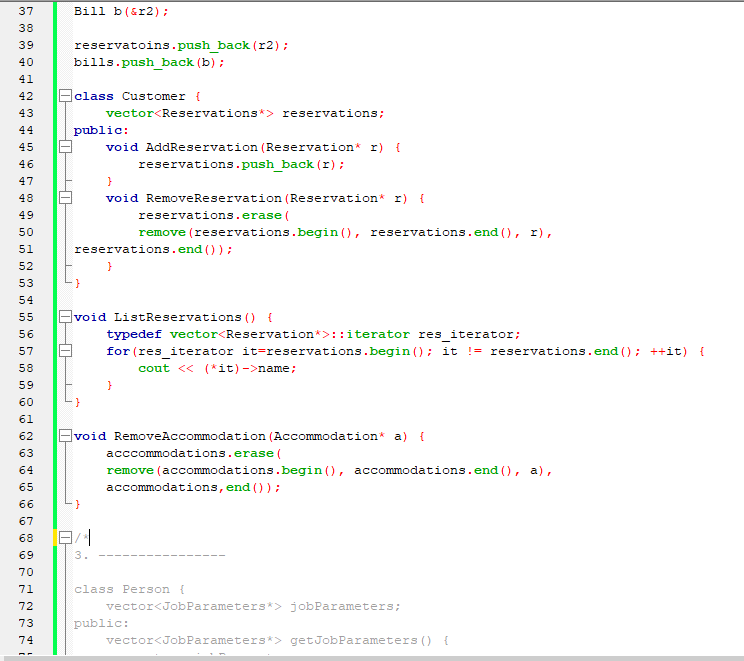
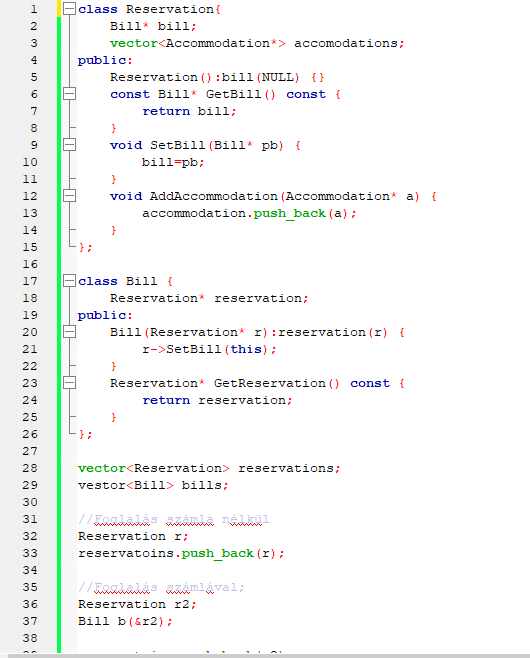
BPMN

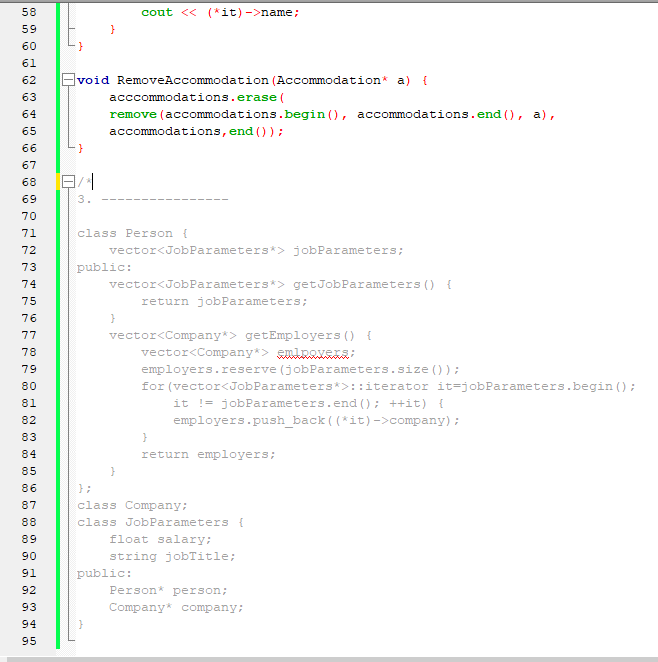
BPEL

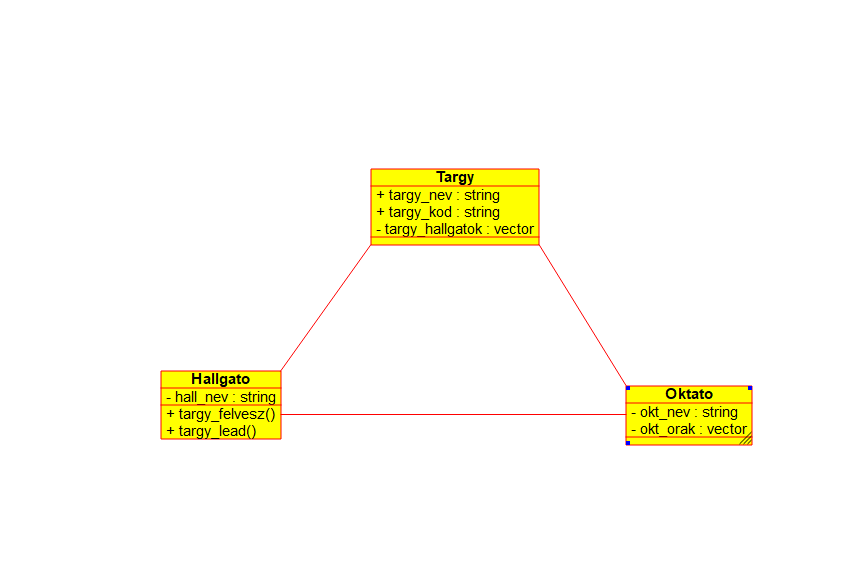
2018.09.29.: Elkezdtem a **Reverse** feladatot. A kód importálás kész. Az eredeti binfa programban csak 1 osztály van ezért a diagram is elég egyszerű.



Az **Esettan** feladatot elkezdtem, a könyvből az első esettanulmányt elolvastam, de a mintafeladatok kódját kigyűjtöttem, de még nem importáltam UML-be.





A **Forward** feladatnak a vázáig jutottam el. Az egyetemi rendszer leegyszerűsített változatát szeretném modellezni. Az oktatók, a hallgatók és a tárgyak kapcsolatát nagy vonalakban készítettem el eddig.  


4. hét

Encoding (tárgyhéten befejezve)

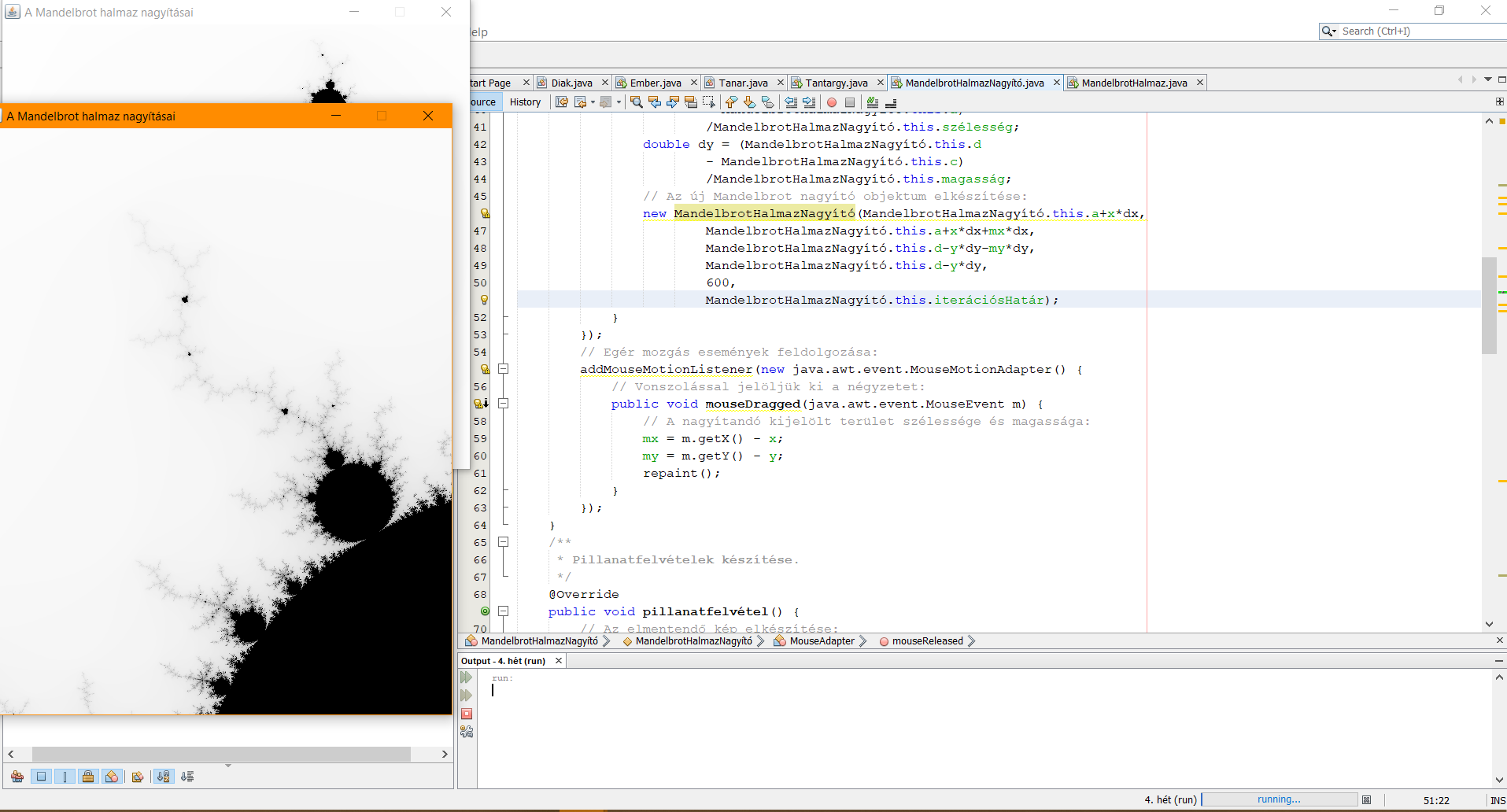
OOCWC lexer

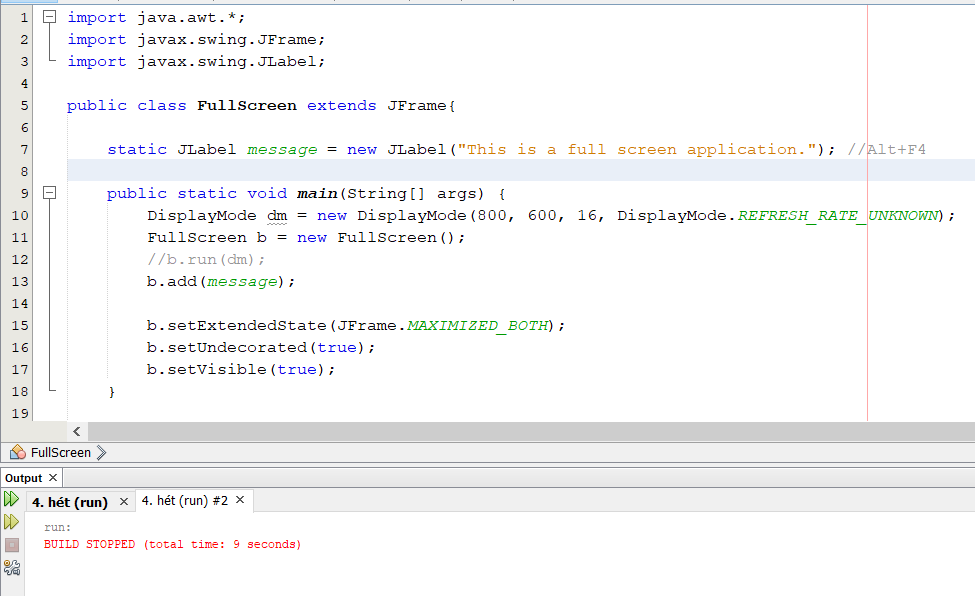
l334d1c4 (tárgyhéten befejezve)

Full screen (tárgyhéten befejezve)

Perceptron osztály

2018.10.03.: Az **Encoding** feladathoz letöltöttem a forráskódokat, majd fordítottam, futtattam a kódokat. Minden probléma nélkül futottak az ékezetes betűkkel is. Ahogyan a programnak kell, nagyítás esetén az új ablakokat megnyitja.  
A nagybetűk használatát azért tehetjük meg probléma nélkül, mert a Java UTF-8 kódolást, illetve Unicode szabványt alkalmaz. Ez magába foglalja sok más mellett az ékezetes betűinket is.



2018.10.07.: **Full Screen** feladat, miszerint írjunk egy Java programot, amely teljes képernyős.   


**l334d1c4** feladat az volt, hogy írjunk egy programot, ami a normál szöveget leet szöveggé alakítja át. A leet szöveg olyan, mint a szleng, nincs rögzített, meghatározott formája. Ezért a leet karakterhelyettesítés az én változatomban:

