Antal János Benjamin NHF
Z80-hoz hasonló assembly futtató

1. Specifikáció

A program egy Z80-as processzor assembly kódjához hasonló, szintaktikailag helyes assembly kódot fog tudni futtatni. Szintaktikai hiba esetén a program csak azt közli majd a felhasználóval, hogy melyik sort találta szintaktikailag hibásnak. A különbségek közül kiemelendő, hogy a program- és az adat memória külön lesz tárolva, így a programmemóriát maga a program nem lesz képes módosítani. Továbbá nem lesz lehetőség a verem beállítására, az automatikusan a memória tetejétől fogja elfoglalni a helyet.

A felhasználó szempontjából érdekes műveletek a következők:

- assembly kód írása
- assembly kód mentése
- assembly kód betöltése
- az egész „processzor” állapotának mentése
- az egész „processzor” állapotának betöltése
- a kód lefordítása
- az lefordított kód futtatása

A felhasználó a program futása során megtekintheti a memória állapotát, azonban azt szerkeszteni csak a parancsokon keresztül tudja. A programnak grafikus felülete lesz.

2. Megoldás rövid szöveges ismertetése

A megoldás során több saját osztályt létrehoztam, melyek közül legfontosabb két osztály az Application és a Processor. Ezek mellett van még a Memory, Register, melyeket a Processor osztály használ. Megemlítendő még a Command abstrakt osztály, mely a különböző parancsoknak a közös őse. A CommandCreater abstrakt osztály felelős a parancsok létrehozásáért. A processzorhoz lehet különböző parancsokat hozzáadni, melyek az executeAll() függvény meghívására egymás után végrehajtódnak. A CommandCreater create() metódusának bővítésével, valamint a megfelelő osztály létrehozásával egyszerűen kibővíthető a processzor parancskészlete. Jelenleg a következő parancsok vannak implementálva:

- **eq** : konstans deklarálását teszi lehetővé. Pl.: „`eq var 14h`”. Ezután a „`var`” szót bárhova írva a hexadecimális 14-et helyettesíti a fordító. A konstansok bárhol használhatók számok ként.
- **ld** : a memóriából egy registerbe, vagy a memóriába betölt egy registert vagy egy konstanst. Az első operandus a hova, a második a mit. A registerek lehetnek A-H és L. A memória való hivatkozás () között történik, lehet szám vagy regiszter. Pl. „`ld (16) A`”. A 16-os memóriacímre betölti az A regiszter tartalmát.
- **add, and, or, pop, push, sub, xor** : A megszokott aritmetikai és logikai műveletek. 1 operandusuk lehet, ami szám vagy regiszter. A másik operandus az „A” regiszter, mint akkumulátor, az eredmény is itt jelenik meg.

- **cmp** : Egy operandusa lehet, szám vagy regiszter, melyet kivon az akkumulátor tartalmából, és az F regiszter 0. bitjét 1-be állítja, ha az eredmény 0. Az akkumulátor tartalma nem változik.
- **jmp, jnz** : Egy operandusuk lehet, az ugrás címe relatívan hivatkozva („-2” = két paranccsal vissza). A jmp mindenképpen ugrik, a jnz pedig akkor, ha az utolsó cmp eredményeként az F regiszter 0. bitje 1-es.
- **inc, dec, not** : Szintén a megszokott aritmetikai és logikai művelet, ellenben operandus nélkül, az akkumulátor tartalmát változtatják meg.

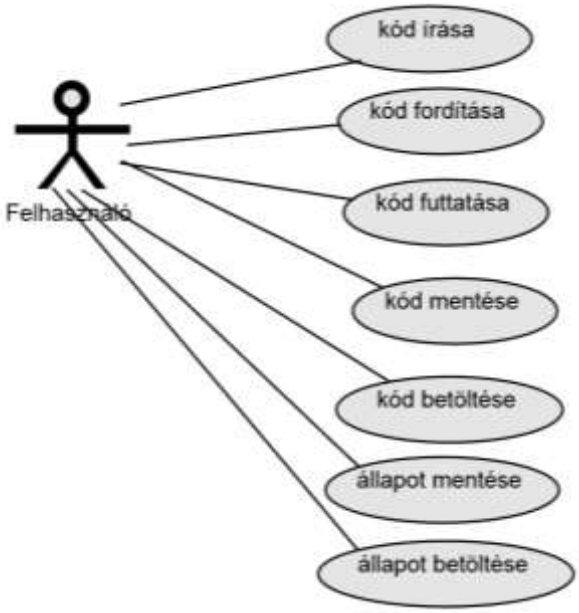
A számok minden esetben lehetnek decimálisak, binárisak vagy hexadecimálisak. Bináris számok végére b-t, hexadecimális számok végére h-t kell írni. A program nem tesz különbséget kis- és nagybetűk között.

Az Application osztály felelős a grafikus megjelenítésért. Tartalmaz 6 darab belső osztályt, melyek ActionListener leszármazottak, az egyes gombokhoz/menüelemekhez vannak hozzáadva. Az osztálynak a konstruktora építi fel az egész grafikus megjelenítést, a menüsávot, a program beviteli mezőt, a gombokat és a memória kijelző részt is.

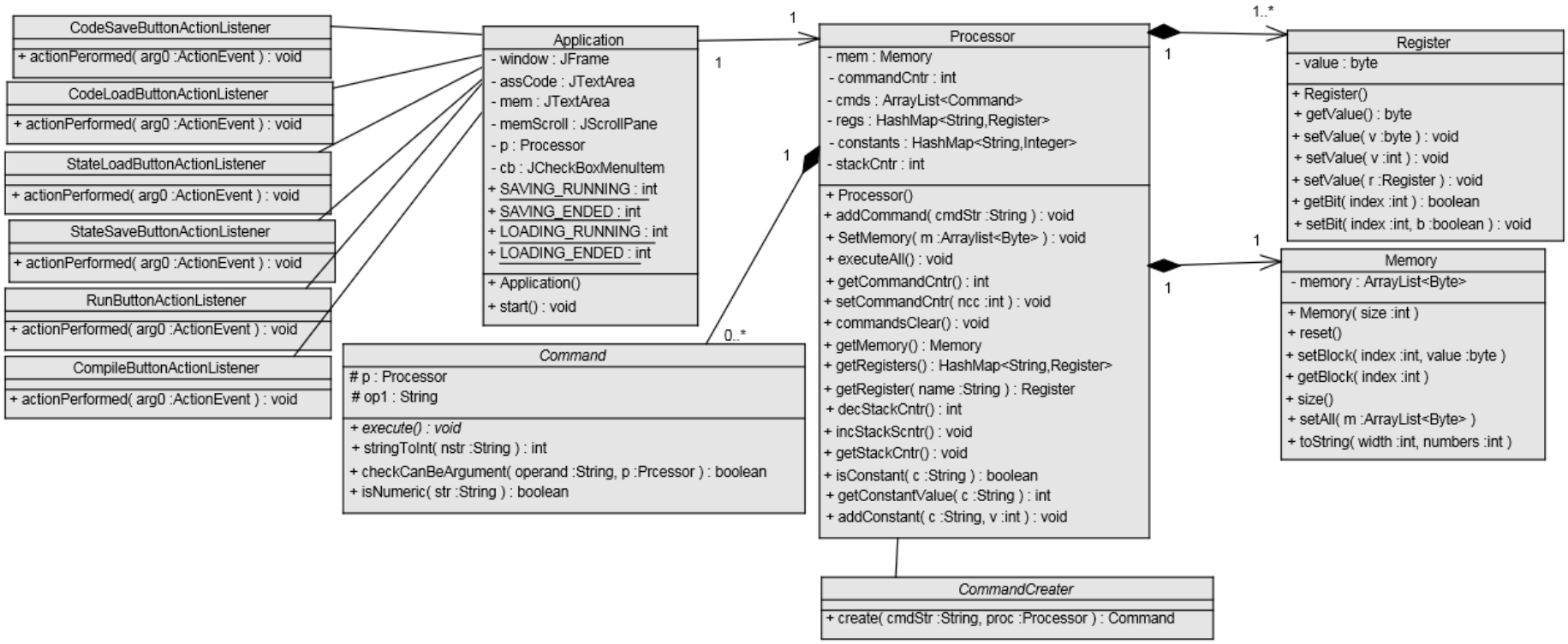
Az osztály diagramot három részben közölném, mivel egyben túl nagy lenne és három részre bontható úgy, hogy elvben ne sérüljön.

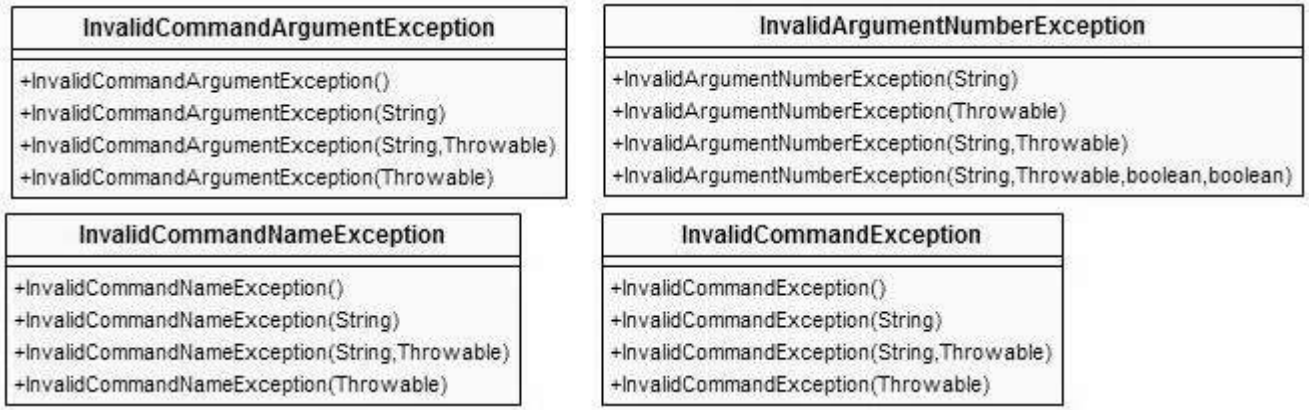
3. Diagramok

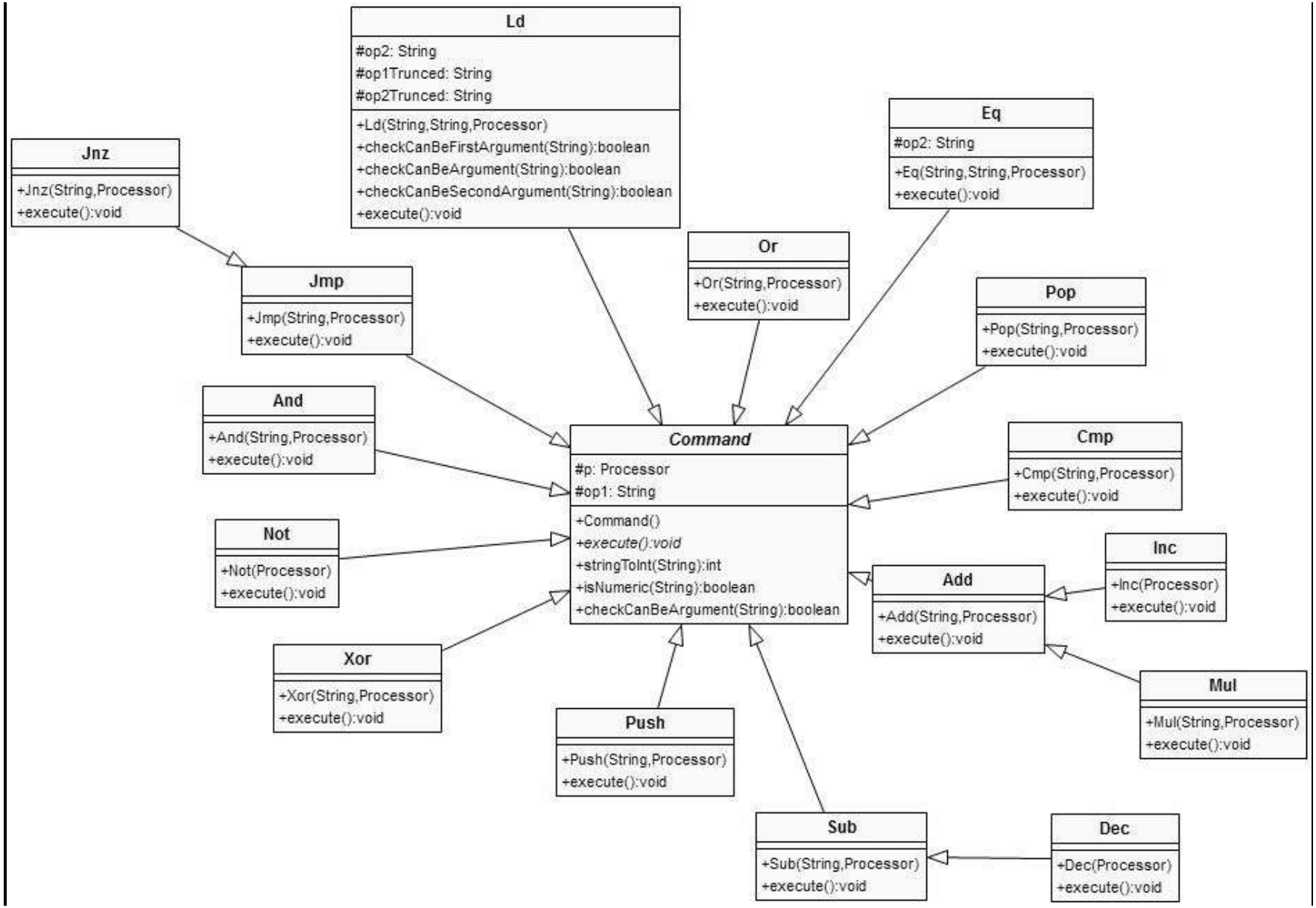
Use Case diagram



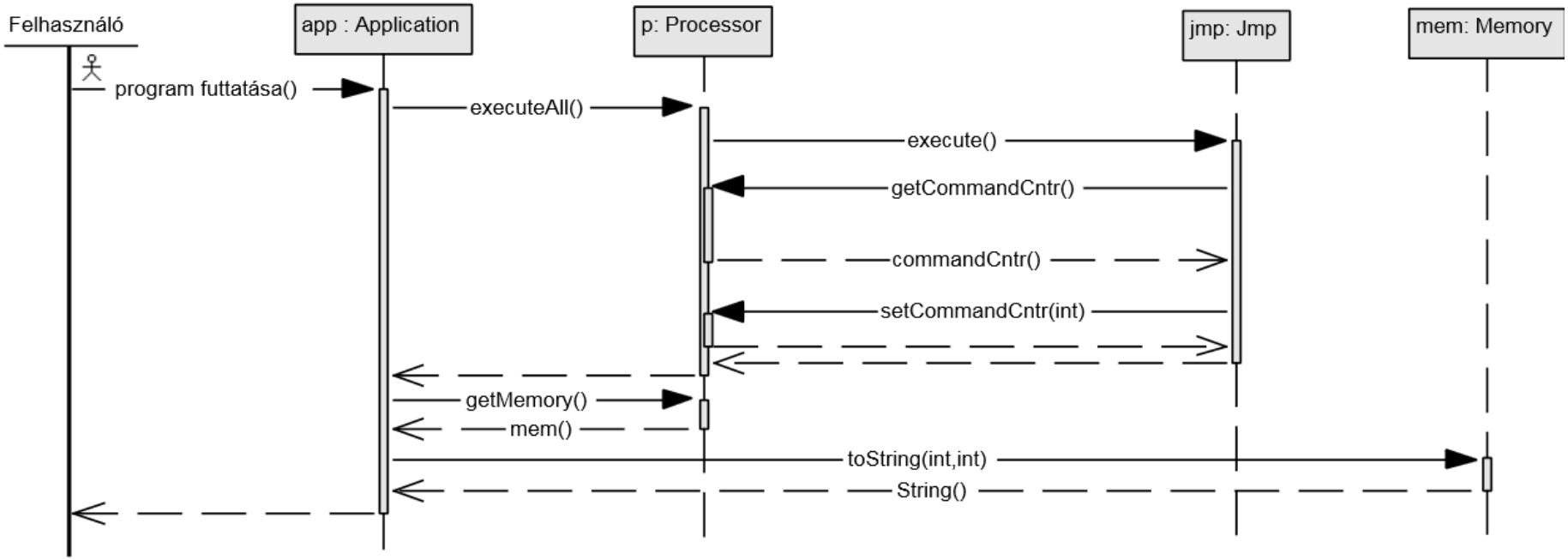
Osztály diagramok

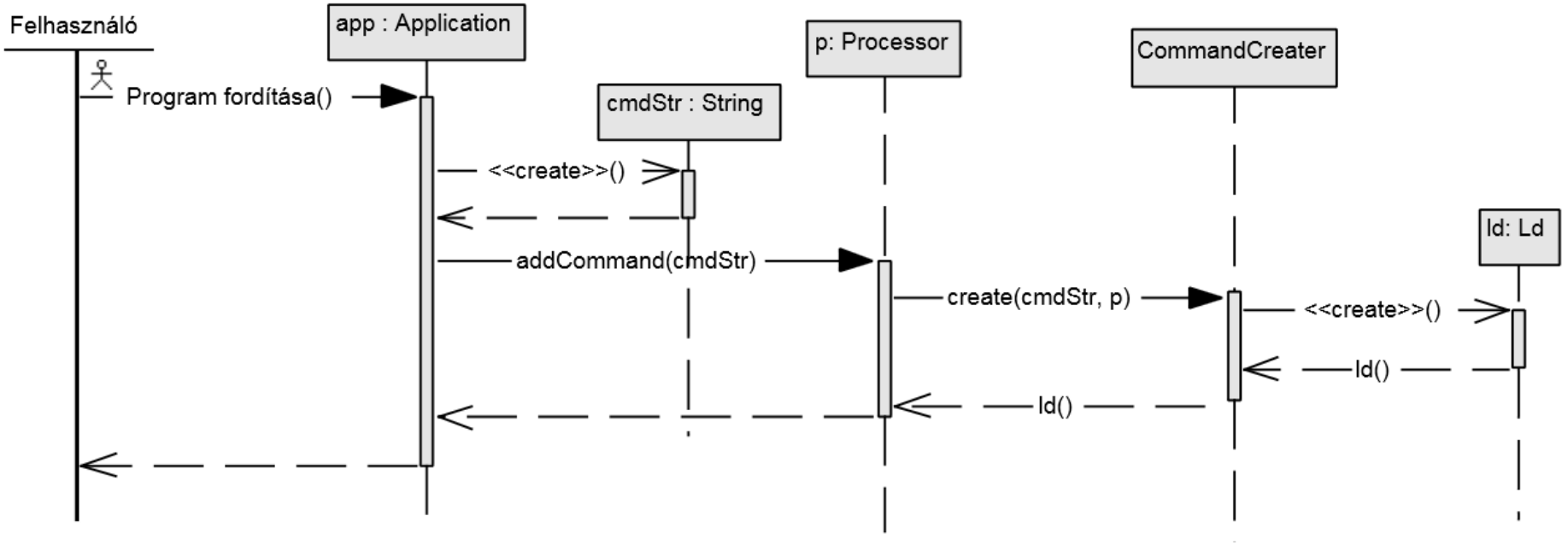



InvalidCommandArgumentException
+InvalidCommandArgumentException()
+InvalidCommandArgumentException(String)
+InvalidCommandArgumentException(String,Throwable)
+InvalidCommandArgumentException(Throwable)
InvalidArgumentNumberException
+InvalidArgumentNumberException(String)
+InvalidArgumentNumberException(Throwable)
+InvalidArgumentNumberException(String,Throwable)
+InvalidArgumentNumberException(String,Throwable,boolean,boolean)
InvalidCommandNameException
+InvalidCommandNameException()
+InvalidCommandNameException(String)
+InvalidCommandNameException(String,Throwable)
+InvalidCommandNameException(Throwable)
InvalidCommandException
+InvalidCommandException()
+InvalidCommandException(String)
+InvalidCommandException(String,Throwable)
+InvalidCommandException(Throwable)


Ld
#op2: String
#op1Trunced: String
#op2Trunced: String
+Ld(String,String,Processor)
+checkCanBeFirstArgument(String):boolean
+checkCanBeArgument(String):boolean
+checkCanBeSecondArgument(String):boolean
+execute():void
Jnz
+Jnz(String,Processor)
+execute():void
Eq
#op2: String
+Eq(String,String,Processor)
+execute():void
Or
+Or(String,Processor)
+execute():void
Jmp
+Jmp(String,Processor)
+execute():void
Pop
+Pop(String,Processor)
+execute():void
And
+And(String,Processor)
+execute():void
Command
#p: Processor
#op1: String
+Command()
+execute():void
+stringToInt(String):int
+isNumeric(String):boolean
+checkCanBeArgument(String):boolean
Cmp
+Cmp(String,Processor)
+execute():void
Not
+Not(Processor)
+execute():void
Inc
+Inc(Processor)
+execute():void
Add
+Add(String,Processor)
+execute():void
Xor
+Xor(String,Processor)
+execute():void
Mul
+Mul(String,Processor)
+execute():void
Push
+Push(String,Processor)
+execute():void
Sub
+Sub(String,Processor)
+execute():void
Dec
+Dec(Processor)
+execute():void

Szekvencia diagramok




Felhasználó
app : Application
p: Processor
CommandCreater
cmdStr : String
ld: Ld
Program fordítása()
<<create>>()
addCommand(cmdStr)
create(cmdStr, p)
<<create>>()
ld()
ld()