

2023/24 őszi félév, Konkurens programozás beadandó

Feltételek

- A beadandó általános feltételei az előadás Canvasében, a Tematika oldalon olvashatók.
- A feladatban megadott neveket betűre pontosan úgy kell használni, ahogy meg vannak adva.
- A megoldás legyen a lehető legjobb minőségű.
 - A Java nyelv szokásos konvencióit követni kell.
 - A kód szerkezete, a változók nevei legyenek megfelelőek.
- Beadás.
 - Az elkészített megoldást **zip** formátumba csomagolva kell feltölteni a Canvasbe.
 - A **zip** csak a megoldás forrásfájljait tartalmazza.
 - Más fájlokat (pl. **.class**) és könyvtárakat ne tartalmazzon.
 - A megoldás a határidőn belül többször is beadható.
 - Az utoljára beadott megoldás kerül értékelésre.
 - *A határidő éles, aki lemarad, az kimarad!*

Alapfeladat (10 pont)

Ebben a feladatban néhány szál segítségével egy egészeket tartalmazó, nyilvános, osztályszintű, **data** nevű tömb adattag elemeit mozgatjuk új pozíciókra. Az alapfeladatban ehhez közvetlenül fogunk szálakat használni.

A névtelen csomagba tartozó **DataMover** program első parancssori paramétere egy közös várakozási idő, a tömbelemek közötti mozgatás szimulált ideje. A többi paraméter egyedi szálak várakozási értékeit adják meg ezredmásodpercben. Ezek darabszámának megfelelő szálakat indítunk el, és a tömbnek is ennyi eleme lesz. A szálakat nullától sorszámozzuk, és a szál objektumaik az osztály nyilvános, osztályszintű, **movers** nevű listájába kerülnek be a létrehozás után.

Ha nem kaptunk egyetlen parancssori paramétert sem, akkor járjunk el úgy, mintha a **123 111 256** paraméterek érkeztek volna.

A tömb elemeit kezdetben a **0**, **1000**, **2000** stb. elemekkel töltjük fel.

Minden szál a következőket teszi **10** ismétlésen át.

1. A sorszámnak megfelelő ideig várakozik.
2. Egy kritikus szakaszban, amelybe a többi szál nem szólhat bele:
 1. A saját sorszámnak megfelelő indexű tömbelemből levonja a saját sorszámat.
 2. Erről üzenetet ír a sztenderd kimenetre, így: **#2: data 2 == 1996**
 3. Kivárja a mozgatási időt.
 4. A saját sorszáma után következő elemet (vagy ha ilyen nincs, az elsőt) megnöveli a saját indexének értékével.
 5. Erről üzenetet ír a sztenderd kimenetre, így: **#2: data 0 -> 14**

A főprogram bevárja mindegyik szál leállítását a program vége előtt. A **movers** listában maradjanak benne a szál objektumok az után is, hogy a szálak elkészültek.

A főprogram végül kiírja a tömb végső tartalmát. Három szál esetén ez a következő: **[20, 990, 1990]**

A feladat akkor tekinthető megoldottnak, a kód jó minőségű és problémák (error/warning) nélkül lefordul és

fut, a leírtaknak megfelelő szerkezetű, és futtatáskor (akár megadtunk paramétereket, akár nem) a program véges időn belül leáll, és a helyes eredményt adja.

Bővített feladat (10 pont)

A névtelen csomagba tartozó `DataMover2` program `n` darab parancssori paramétert kap, ennyi szál fog egymás között adatokat átadni több lépésben.

Az alábbi adattagok legyenek mind nyilvánosak és osztályszintűek:

- `arrivalCount`, `totalSent`, `totalArrived`: három `AtomicInteger`, nulláról indulnak
- `pool`: egy `ExecutorService`
- `queues`: egészeket szállító `BlockingQueue` adatszerkezetek listája
- `moverResults`: számított `DataMover2Result` értékek `Future` tárolóinak listája
 - A `DataMover2Result` osztály három, nyilvánosan elérhető egészét tárol: `count`, `data`, `forwarded`
- `discards`: egészek listája

Ha nem kaptunk egyetlen parancssori paramétert sem, ismét járunk el úgy, mintha a `123 111 256 404` paraméterek érkeztek volna.

Induláskor a főprogram feltölti a `queues` listát `n` elemmel, és készít egy legfeljebb `100` méretű `pool` t.

A következő részben `n` feladatot adunk a `pool` nak, ezek mindegyike egy `DataMover2Result` értéket számít ki a tevékenysége során, és azt adja vissza. Nevezzük ezt `result` nak.

- Tekintsük úgy, hogy a `queues` lista `i`-edik eleme az `i`-edik és az azt követő feladatot köti össze.
- A feladatok addig működnek, amíg összességében `5n` érték be nem érkezett a helyére. Mindaddig a következőket ismétlik:
 - `sends <x>`: Előállítanak egy `0` és `10_000` közötti `x` értéket, és blokkoló módon a tőlük kifelé vivő `queue` ba helyezik.
 - Ezt adjuk hozzá a `totalSent` változóhoz.
 - A befelé jövő `queue` ból `300..1000` ezredmásodperc közötti, véletlenszerűen választott ideig próbálnak egy értéket fogadni.
 - `got nothing...`: Ha nem jött érték, a feladat rögtön új iterációt kezd meg.
 - `got <x>`: Ha az érték `n` nel vett maradéka éppen a feladat sorszáma, akkor az érték beérkezett a helyére.
 - Nőjön meg `arrivalCount` és `result.count` értéke eggyel, `result.data` pedig a bejött értékkel.
 - `forwards <x>`: Különben a feladat küldje tovább az `x-1` értéket blokkoló küldéssel.
 - Nőjön meg `result.forwarded` értéke eggyel.
 - A feladat a parancssorban megkapott ideig várakozik.
- A kiszámított `result` okat hamarosan felhasználjuk, ezeket a `moverResults` fogja közvetíteni.

A szálak az értékek küldéséről és fogadásáról üzeneteket írnak a sztenderd kimenetre, ilyen stílusban:

```
...
total  9/20 | #0 got 7628
total  9/20 | #1 forwards 2426 [2]
total 10/20 | #1 sends 9281
total 10/20 | #3 got nothing...
...
```

Itt az első két szám az eddigi beérkezéseket és az elvárt összes beérkezést mutatja, a `#m` a kiírást végző

feladat sorszáma, a `[v]` érték pedig azt mutatja, melyik sorszámú feladat fogadná el az értéket.

A főprogram, amint elindultak a feladatok, jelezze a `pool` felé, hogy az ne fogadjon innentől több feladatot, és adjon neki legfeljebb `30` másodpercet, hogy a számítások terminálhassanak.

Amikor minden számítás elkészült, az eredményeiket vegyük fel a `moverResults` lista elemei segítségével, és a `result.data` és `result.forwarded` értékeket adjuk össze a `totalArrived` tárolóba. Mivel a `queues` elemeiben még maradhattak számok, ezeket összegezzük a `discards` listába.

Végül összegezzük a kapott tartalmakat, és vessük össze a kétféle módon kijött eredményt:

a `totalSent` érték meg kell, hogy egyezzen a `totalArrived` és a `discards` összegével. Egy lehetséges lefutás vége:

```
total 18/20 | #1 got 9341
total 19/20 | #1 sends 2386
total 19/20 | #0 sends 9761
total 19/20 | #1 got 9761
total 19/20 | #0 forwards 2771 [3]
discarded [0, 2770, 193509, 58277] = 254556
sent 370278 == got 370278 = 115722 + discarded 254556
```

Ha esetleg rossz eredmény jönne ki, azt az alábbi módon lehet jelezni. Erre teljes pontot érő megoldásban nem kerül rá a vezérlés.

```
WRONG sent 370278 != got 152 = 24 + discarded 128
```