2023/24 őszi félév, Haladó Java 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 megfelelők.
- Beadás.
  - Az elkészített megoldást **zip** formátumba csomagolva kell feltölteni a Canvasbe.
  - A `zip` csak a megoldás forrásfájljait tartalmazza a megfelelő könyvtárstruktúrában.
  - Más fájlokat és könyvtárakat (pl. `.class`, IDE projekt) ne tartalmazzon.
  - A megoldás a határidőn belül többször is beadható.
    - Az utoljára beadott megoldás kerül értékelésre.
    - *A határidő éles, aki lemarad, az kimarad!*

Alapfeladat (7 pont)

Ebben a feladatban a `task.compulsory.MultiProducer.multiProducerFactory` nyilvános láthatóságú, osztályszintű adattagba olyan („külső”) lambdát kell elkészíteni, ami egy másik („belső”, a továbbiakban `multiProducer`, de ennek a kódban nem feltétlenül jelenik meg a neve) lambdát ad ki.

A belső lambda két paramétert vár, ezek szintén lambdák. Az első paraméter (`amountLambda`) `int` értékeket szolgáltat (nem `Integer`eket), a második (`contentLambda`) pedig szövegeket. A `multiProducer` célja az, hogy a `contentLambda` tól kapott szövegeket bocsássa ki, egy megkapott szövegből egymás után annyit, amilyen számot az `amountLambda` ad.

- Az `amountLambda` adhat nekünk nullát vagy negatív számot is. Ekkor el kell vetni a másik lambdából éppen megkapott szöveget, és újat kell kérni abból is, és számból is.
  - Az újonnan kapott szám is lehet nempozitív...
- Példa: ha az `amountLambda` sorban a `3, 6, -1, -2, 0, 2, ...` számokat szolgáltatja, a `contentLambda` pedig sorban az `a, b, c, ...` szövegeket, akkor a `multiProducer` ismételt meghívásai sorban a következő kimeneteket adják: `a, a, a, b, b, b, b, b, b, f, f, ...`

A külső lambda nem kap paramétert, a feladata csupán azt biztosítani, hogy egyszerre több, egymástól független `multiProducer` jellegű lambdánk lehessen.

Tesztelés

A `task.test.MultiProducerTest` tesztelje JUnit 5 segítségével a fentieket a következőképpen.

- Legyen négy nyilvános, példányszintű lambda adattagja. Kettő ezek közül (`amountLambda123A` és `amountLambda123B`) a `0, 1, 2, 3, ...` értékeket adja ki egymástól függetlenül, a másik kettő (`contentLambdaA` és `contentLambdaB`) pedig az `a, aa, aaa, aaaa, aaaaa, ...` szövegeket adja ki.
  - Az állapotaikat a `value123X` és a `txtX` nevű, nyilvános, példányszintű adattagok tárolják egyelemű tömbökben, ahol `X` az egyik `A` és `B` közül.
- A `task.test.MultiProducerTest.test` tesztelő metódus használja fel az említett lambdákat az egyetlen logikus kombinációban, és a két adódó lambdát hajtsa meg hét lépés erejéig, váltakozó sorrendben.
  - Tehát egy lépést tesz meg a megkapott `A` lambda, aztán egyet a `B`, aztán megint egyet az `A` stb.
  - Ellenőrzendő, hogy a kapott tartalmak egy-egy lambdára a következők, és a lambdák függetlenek, egymást nem zavarják: `a`, `aa`, `aa`, `aaa`, `aaa`, `aaa`, `aaaa`.
  - Az alapfeladat megoldása során szabad folyamokat használni, de nem kötelező.

A feladat akkor tekinthető megoldottnak, a kód jó minőségű és problémák (error/warning) nélkül lefordul és fut, a leírtaknak megfelelő szerkezetű, és a teszt helyes eredményt ad.

Bővített feladat (8 pont)

A `task.extension.MultiProducer2.cachedMultiProducer` nyilvános láthatóságú, osztályszintű adattag lambdája az előző feladat egy változata lesz. Abban a feladatban a `contentLambda` egymástól független szövegeket adott át nekünk, és azokból adtunk ki néhány példányt, most viszont a kimeneti szövegek egymásból alakulnak ki. A könnyebb érthetőség kedvéért fordított sorrendben, a végeredmény szerkezetét ismertetve lássuk, mi történik.

A legbelső lambdánk (`multiProducer`) az őt érő (paraméter nélküli) hívásokra sorban szövegeket ad ki, ezeket jelölje `txt1, txt2, txt3, txt4, txt5, txt6, txt7, ...`

Az első ezek közül az üres szöveg. A következő úgy adódik, hogy egy `decisionLambda` lambda eldönti, hogy a következő kiadandó szövegnek meg kell-e egyeznie az előzővel, vagy bővülnie kell-e: a `task.extension.MultiProducer2State` felsorolási típus `KEEP` értékét adja nekünk az előbbi, az `EXTEND` értékét adja az utóbbi esetben. Bővüléskor az `appendTxt` szöveget fűzzük a korábbihoz.

- Példa: ha `appendTxt` egyetlen `a` betűt tartalmaz, és a `decisionLambda` a `KEEP, KEEP, EXTEND, EXTEND, EXTEND, EXTEND, KEEP, KEEP, EXTEND, ...` döntéseket hozza, akkor a kijövő elemek sorban `<üres>` , `<üres>` , `<üres>` , `a` , `aa` , `aaa` , `aaaa` , `aaaa` , `aaaa` , `aaaaa` , ...
- Azért három szöveg üres kezdetben, mert a kezdeti szöveget is kiadjuk egyszer, még mielőtt akár egyetlen döntést is hozna a `decisionLambda` .

Bonyodalom: a `decisionLambda` eszközről ismert, hogy érzékeny, és gyors egymásutánban legfeljebb `decisionCount` (ez egy pozitív szám) döntést szabad meghozatni vele. Ezért felveszünk egy gyorsítótárat (`cache`), ami egy lista, és kezdetben az üres szöveget tartalmazza. Amikor a gyorsítótár egy elemet tartalmaz (pl. a legelején), akkor elkészítjük bele a következő `decisionCount` elemet.

- Tehát etapokban dolgozunk: feltöltjük a gyorsítótárat `decisionCount` elemmel, majd lefogyasztjuk.
- Példa: ha `decisionCount` értéke `3` , akkor a korábbi példa a következőképp működik:
  i. A gyorsítótárban az üres szöveg az egyedüli elem.
  ii. Amikor a `multiProducer` lambdát először hívják meg, érzékeljük, hogy majdnem kifogyott.
    a. Feltöltjük ezekkel: `1. <üres>, 2. <üres>, 3. <üres>`
      - A számok csak az olvasást könnyítik, nem jelennek meg a kódban.
      - A feltöltéshez kétszer meg kellett hívni a `decisionLambda` t.
    b. Kiadjuk az első elemet.
      - Ezt természetesen ki is vesszük a gyorsítótárból.
  iii. Amikor a `multiProducer` lambdát másodszor hívják meg, kiadjuk a `2.` elemet.
  iv. Újabb hívásnál érzékeljük, hogy ismét kevés az elem, és ismét feltöltjük a gyorsítótárat.
- Fontos: a legutoljára elkészített elemből kiindulva kell a folytatást elkészíteni, de azt az elemet még egyszer nem kell betenni a gyorsítótárba!

A fentiek alapján a `cachedMultiProducer` technikai részletei.

- A külső lambda megkapja a `decisionCount` (egész, feltételezhető, hogy pozitív) és a `decisionLambda` paramétereket. Utóbbi `MultiProducer2State` értékeket kiadó, paraméterek nélküli lambda.
- A külső lambda kódja tartalmazza a szövegeket tároló `cache` listát. Ez kezdetben egyetlen elemmel, egy üres szöveggel van feltöltve.
- A külső lambda visszatérése olyan lambda, ami az `appendTxt` szöveget veszi át, és egy paraméter nélküli lambdát ad vissza.
  - Ez a legbelső lambda megvizsgálja, hogy `cache` mérete egy-e.
    - Ha igen, akkor veszi ezt az elemet, és összesen `decisionCount` lépésen keresztül elkészíti a `cache` új elemeit belőle.
    - Az új elem így jön ki a megelőzőből: meghívjuk a `decisionLambda` t. Ha ennek az eredménye `KEEP` , akkor az új elem tartalma egyszerűen a megelőzőé, ha pedig `EXTEND` , akkor a megelőző végéhez hozzáfűzve az `appendTxt` szöveget.
    - Ezt a részét a feladatnak *kötelező egyetlen **Stream** használatával megoldani*. Más eszközökkel elkészített megoldás akkor sem ér itt pontot, ha egyébként teljesen helyes.
  - Végül a (legbelső) lambda kiveszi a `cache` első elemét, és visszatér vele.
- Itt is és a tesztelőben is `static import` stílusban használjuk a `MultiProducer2State` elemeit.

Az osztályban legyen továbbá egy `oneFromEach` metódus, amelynek egy `T` sablonparamétere van, nincsen (érték) paramétere. Visszatérése egy olyan lambda, ami paraméterként két olyan lambdát vár, amelyek `T` típusú elemeket adnak ki, és eredménye egy olyan lambda, amely felváltott sorrendben adja ki ezeket az elemeket.

- Példa: ha az egyik bemeneti lambda az `a, b, c, ...` , a másik pedig az `1, 2, 3, ...` szövegeket adja ki, akkor az eredmény lambda sorban az `a, 1, b, 2, c, 3, ...` szövegeket adja ki.

Tesztelés

A `task.test.MultiProducerTest2.testConstant` metódus paraméterezett tesztelővel próbálja ki, hogy ha a `cachedMultiProducer` olyan `decisionLambda` t kap, ami folyton a `KEEP` értéket állítja elő, akkor az így kapott lambda mindig az üres szöveggel tér vissza.

- Paraméterek:
  - a kapott lambdának átadandó szöveg (tetszőleges lehet, akár üres is)
  - `decisionCount` (legyen kis és nagy érték is közte)
  - hanyadik hívás után vizsgáljuk meg, hogy üres szöveg-e az eredmény
- Ilyen jellegű paraméterezésből három-négy teszt fusson le.

A tesztelő osztály `setup` metódusa minden teszteset lefutása előtt állítson be két adattagot.

- `flipDecisionLambda` : sorban az `EXTEND, KEEP, EXTEND, KEEP, ...` értékeket állítja elő a végtelenségig
- `bunchDecisionLambda` : sorban az `EXTEND, KEEP, EXTEND, KEEP, KEEP, EXTEND, KEEP, KEEP, KEEP, EXTEND, ...` értékeket állítja elő a végtelenségig
  - Tehát mindig eggyel több `KEEP` kerül az `EXTEND` értékek közé.

A tesztelő osztály `flip20` és `bunch20` metódusa tesztelje a lambdákról, hogy az első 20 kijövő érték megfelelő-e.

A `task.test.MultiProducerTest2.oneOfAB` metódus próbálja ki a `oneFromEach` metódust olyan lambdákkal, amelyikek csupa `"a"` és csupa `"b"` értéket adnak. Az első hat adódó értéket a tesztelő metódus gyűjtse össze egy `Stream` segítségével egy listába. Ennek elvárt tartalma `"a", "b", "a", "b", "a", "b"` .

A `task.test.MultiProducerTest2.cachedMultiProducerTest` metódus próbálja ki a `cachedMultiProducer` t: készítsen a `flipDecisionLambda` és a `bunchDecisionLambda` felhasználásával is egy-egy lambdát. Az elsőnek az `"a"` , a másodiknak a `"b"` szöveget adjuk át; a `decisionCount` minden esetben a tesztelő metódus paraméteréből jöjjön, ahol adjunk meg néhány értéket (pl. 1, 3, 10). A tesztelő a `oneFromEach` segítségével hajtsa ezeket meg felváltva, és egy `Stream` segítségével vegyük az első 20-20 elemet (összesen negyvenet), amelyeket gyűjtsünk össze egy listába. Ennek az elvárt elemei közül az első kettő üres, majd a továbbiak:

```
a,          b,
a,          b,
aa,         bb,
aa,         bb,
aaa,        bb,
aaa,        bbb,
aaaa,       bbb,
aaaa,       bbb,
aaaaa,      bbb,
aaaaa,      bbbb,
aaaaaa,     bbbb,
aaaaaa,     bbbb,
aaaaaaa,    bbbb,
aaaaaaa,    bbbb,
aaaaaaaa,   bbbbb,
aaaaaaaa,   bbbbb,
aaaaaaaaa,  bbbbb,
aaaaaaaaa,  bbbbb,
aaaaaaaaaa, bbbbb,
```