Gyakorló feladatsor

2018 2. Zh pót:
Feladat

Készíts egy osztályt, ami lehetőséget ad erőforrás lefoglalására. Az osztálynak két metódusa van a `lock()` és az `unlock()`.
Ha egy szál meghívja a `lock()`-ot az akkor tér vissza, ha megszerezte a futás jogát (azaz másik szál még nem szerezte azt meg előtte, amennyiben már egy másik szál lefoglalta a lock-ot akkor a szál várakozik míg az fel nem szabadul), a `lock()` azonnal visszatér ha a meghívó szál már birtokolja a lock-ot (ennek eldöntésében segíthet a `Thread.currentThread()` függvény).
Ha egy szál meghívja az `unlock()` függvényt, akkor ha a meghívó szál birtokolja a lock-ot a szál felszabadítja azt és erről értesíti a várakozó szálakat. Ha nem a lock-ot birtokló szál hívja meg az `unlcok()` metódust, akkor IllegalMonitorStateException-t dob.

Teszteléshez indíts 5 szálat amelyek mind 1 lock megszerzéséért versengenek. Amíg birtokolják a lock-ot 5 másodpercet sleepelnek majd közlik, hogy a “<sorszám>. szál végzett”, ezután elengedik a lockot.

Ahhoz, hogy az implementációd pozitív pontszámot érjen, egyáltalán ne használd a `java.util.concurrent` csomag osztályait! A feladat megoldása során figyelj oda, hogy a programod szálbiztos legyen, valamint ne használj se busy waitinget se pollingot. Ügyelj arra, hogy ne alakulhasson ki deadlock vagy livelock valamint - a lehetőségekhez mérten - valódi párhuzamosság legyen benne.

2019 2. ZH:
1. Feladat

A feladat egy street food stand szimulációja - szakáccsal, asztalokkal és vendégekkel. Az érkező vendégek az első szabad asztalhoz leülnek, és leadják a rendelésüket. A standban egy szakács van, akinek egy ételt elkészíteni egy másodpercig tart. Az elkészült ételeket a vendég 2 másodpercig fogyasztja (+/- fél másodperc véletlen idő), majd utána rendel még egy ételt, aminek hasonló elfogyasztása után távozik. A standnál négy asztal van, az este során harminc vendég érkezik, átlag egy másodperces (+/- fél másodperc véletlen) időközökkel.

- A szakács, és a vendégek mind külön szálon fussanak
- A négy asztal mindegyikénél csak egy vendég ülhet
- A rendelések pillanatszerűek, akármennyi rendelés felkerülhet egy rendelés-listára
- A szakács egyszerre csak egy ételt tud főzni, ha elkészült a kész ételt egy elkészült listába teszi, ahonnan a vendég elveheti a sajátját (és csakis a sajátját!)
- Semelyik résztvevő ne foglaljon feleslegesen erőforrásokat.
- Ne használjon busy-wait vagy polling módszereket a várakozásra
- Logolja az eseményeket a konzolra (érkezés, asztal elfoglalása, rendelés leadás, főzés, étkezés, távozás)

2. Feladat

Egy számológép programot kell elkészíteni, alapvető funkcionalitással a legegyszerűbb módon. A számológép, az egyszerűség és időmegtakarítás céljából - stílusosan - egy bináris számológép legyen, ahol a számjegyek a 0 és az 1 legyenek.
A számológépnek legyen tehát:

- 0 és 1 gombja
- + és - gombja (a számítás idején ne lehessen inputot bevinni)
- = az eredmény számításához (a számítás idején ne lehessen inputot bevinni)
- <- egy számjegy törléséhez
- C a kijelző törléséhez
- Szöveges mezője, ahol a bevitt számok és az eredménye jelenik meg

A számológépet a JavaFX eszközeivel kell elkészíteni, az egyes nyomógombok egérkattintásra, valamint a billentyűzeten Enter nyomására is működjenek.
Mivel ezen matematikai műveletek azonnal végrehajtódnak, a számítások lépéséhez fél-fél másodperc lassítást kérünk betenni, és ennek megfelelően reszponzív GUI-t készíteni!

A számítás módja során érdemes az Integer.parseInt(int, int) és a Integer.toBinaryString(int i) függvényeket.

Első feladat

Minden Java ZH-ra ülésrendet szoktunk nyomtatni, azonban ez egy fáradtságos munka, és sok időt elvesz az érdekes feladatok kitalálásától. Szeretnénk ezért megkérni, hogy írj egy géptermi helyfoglaló rendszert!
Az alkalmazás egy szerverből és a hallgatók által használt kliensből áll. A kliens induláskor csatlakozik a szerverre (amely jelen esetben a localhoston egy tetszőleges porton figyel), majd megjelenít egy ablakot, amelyen a lehetséges helyek vannak feltüntetve, valamint alul egy állapotjelző felirat.
A lehetséges helyek gombok legyenek, 5 sorban és 8 oszlopban, az egyes helyeknek megfelelően. A feliratuk jelezze a pozíciójukat, a felirat színe pedig a foglalási állapotot: piros a foglalt és zöld a még szabad hely. A szerver csatlakozáskor elküldi a már lefoglalt helyeket.
A foglalás folyamata a következő: a felhasználó kiválaszt egy helyet, az ennek megfelelő gombot megnyomja. Ha ez nem zöld, akkor az állapotjelző felirat értesít az érvénytelen állapotról. Ha zöld, akkor a program elküldi a szervernek a választást.
Ha a hely még mindig szabad (vigyázz rá, hogy egyszerre többen is használhatják a szolgáltatást), akkor a szerver nyugtázza, majd elküldi a terem jelenlegi állását (amely már nyilván tartalmazza a felhasználó foglalását is). A kliens ez alapján frissíti az ablak tartalmát.
Ha a hely már nem szabad, akkor a szerver ezt közli a klienssel, majd elküldi a terem jelenlegi foglalási állását. A kliens ez alapján frissíti az ablak tartalmát és közli az állapotjelzőn keresztül a klienssel, hogy a foglalás sikertelen.
Akár sikeres, akár sikertelen volt a foglalás, újra lehet próbálkozni.
A javítás során mi is teszteljük, kipróbáljuk a programot, így a maximális pontszám eléréséhez fontos, hogy a tanult technikák használatával és stabilan működjön az alkalmazás.

Második feladat

Fontos, hogy a ZH-t mindenki egyszerre kezdje el, amíg nem készült el mindenki, addig a többiek várnak. Készíts egy Java osztályt, amelyik képes ennek a megvalósítására. (Ez a feladat különbözik az első feladattól, csupán a körítő mese az, amely összeköti.)
Az osztály konstruktora egy számot vár, ez a megjelent hallgatók *N* létszáma.
Egyetlen metódusa van, az `await()`, amelyik addig nem tér vissza, amíg az objektum `await()` metódusát nem hívják meg *N*-szer. Ekkor az összes metódushívás visszatér.
Tehát: az `await()` metódusok hívásakor a megfelelő szálak futása megáll, egészen addig, amíg az *N*-ik szál meg nem hívta az `await()` metódusát az objektumnak. Ekkor az összes szál futása folytatódik tovább.
Minden további hívásra az `await()` metódus rögtön visszatér. Ha az implementáció által használt metódusok valamelyike InterruptedException-t dob, akkor az ne legyen elkapva az `await()`-en belül.
Ugyan mi is látjuk, hogy a kért osztály működése nagyon hasonlít a tanult `java.util.concurrent.CountDownLatch`-éhez, viszont ahhoz, hogy az implementációd pozitív pontszámot érjen, egyáltalán ne használd a `java.util.concurrent` csomag osztályait! A feladat megoldása során figyelj oda, hogy a

programod szálbiztos legyen, valamint ne használj se busy waitinget se pollingot. Ügyelj arra, hogy ne alakulhasson ki deadlock vagy livelock valamint - a lehetőségekhez mérten - valódi párhuzamosság legyen benne.
Segítségül írtunk egy keretrendszert, amelynek `CountDownLatch` osztályát kell implementálnod. A keretrendszer `Main` osztályát futtatva több teszteseten keresztül próbálja ki az implementációdat, kiírva, hogy melyik szál esetén nem működött az `await()` hívás. Ha ez nem jelez ki hibát, az nem jelenti azt, hogy nincs is; ezt mi a kódot átnézve fogjuk eldönteni.