HATÉKONY JAVA PROGRAMOZÁS

5. forduló

MSCI

A kategória támogatója: MSCI

Ismertető a feladathoz

Felhasznált idő: 00:00/15:00 **Elért pontszám:** 0/15

1. feladat 0/1 pont

A felsorolt opciók közül melyik egy valós Java garbage collector implementáció HotSpot JVM esetén?

Válaszok

- [ ] CMC - Concurrent Mark Clean, egy többszálú garbage collector. Olyan alkalmazásoknak javasolt, ahol bár minden alkalmazás szál meg lesz szakítva GC alatt, de a többszálúsag miatt a megszakítás aránylag rövid lehet.
- [x] Serial Garbage Collector - a legegyszerűbb egyszálú garbage collector. Minden alkalmazás szál meg lesz állítva a teljes GC idejére ezen collector futása alatt.
  Ez a válasz helyes, de nem jelölted meg.
- [x] G1 - Garbage First GC. Nagy memóriaterületet foglaló, több processzoros rendszereken futó alkalmazásoknak ajanlott GC.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Parallel G1 Collector - A G1 collector újabb, többszálú megvalósítása.

Magyarázat

Csak a megjelöltek helyesek, a másik kettő nem valós GarbageCollector.

2. feladat 0/3 pont

Adott az alábbi rekurzív algoritmus az n. Fibonacci szám kiszámolására. Melyik állítás igaz az alabbiak közül?

```
public static int fib(int n, int a, int b ) {
  if (n == 0)
    return a;
  if (n == 1)
    return b;
```

```
    return fib(n - 1, b, a + b);
}
```

Válaszok

- [x] Ha n túl nagy, **StackOverflowError**-t fog dobni a függveny.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Ha n túl nagy, **OutOfMemoryError**-t fog dobni a függveny.
- [x] A rekurzió miatt a **stack**-et még n-1-szer fogjuk növelni az első fib hívás után.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A **stack**-et nem kell növelni a folytatólagos fib hívások alatt.
- [ ] Mindegyik korábbi állítás hamis.

Magyarázat

A farokrekurzió miatt más nyelvekben a stack nem nőne, de a Java fordító ezt az optimalizációt még nem tartalmazza. Ezért a **stack**-et növleni kell **n-1**-szer és nagy **n** esetén **StackOverFlow** is előfordulhat.

3. feladat 0/3 pont

Java 11, HotSpot JVM. Az alábbi lehetőségek közül melyik esetben mondhatjuk, hogy egy heap-en található objektum példány felszabadítható a GarbageCollector által?

Válaszok

- [x] A példányra nem mutat referencia, minden korábbi a példányra mutató referencia el lett dobva.
  Ez a válasz helyes, de nem jelölted meg.
- [x] A példány nem érhető el futó/blokkolt/várakozó szálakból
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A példányra ugyan van referencia, de azt sosem olvassuk. (pl. private példány változo és belső metódus nem olvassa, csak írja) Tekintsünk el attól, hogy van-e ennek gyakorlati értelme.
- [ ] A példányra csak gyenge referenciák hivatkoznak, mint olyan inner class-ok implicit outer class-ra mutató referenciái. Az inner class példányai maguk meg nem felszabadíthatóak.

Magyarázat

A példányra ugyan van referencia, de azt sosem olvassuk. (pl. private példány változo és belső metódus nem olvassa, csak írja) Tekintsünk el attól, hogy van-e ennek gyakorlati értelme.:

esetben nem szabadítható fel, akkor sem ha nem fogjuk olvasni soha.

A példányra csak gyenge referenciák hivatkoznak, mint olyan inner class-ok implicit outer class-ra mutató referenciái. Az inner class példányai maguk meg nem felszabadíthatóak.: esetben a referencia nem gyenge referencia.

4. feladat 0/4 pont

Melyik állítás helyes a stack és heap memóriaterületekkel kapcsolatban Javaban?

Válasz

- [ ] A:

  Stack memóriát foglalunk le és szabadítunk fel, ahogy a szálakon függvényhívásokat végzünk. Egy rekurzív függvény emiatt kifuthat a stackből, de a method inlining JIT optimalizáció segítségével ez megelőzhető. Hátránya, hogy a program indításakor már meg kell adni a -XX:MaxInlineSize paramétert, de ha ezt elég nagyra állítjuk, a **StackOverflowError** teljesen elkerülhető.

- [ ] B:

  Amikor a garbage collector a stack memóriát szabadítja fel, megállítja a metódus hívásokat (stop the world GC) hogy ne nőjjön/csökkenjen a stack a GC alatt.

- [ ] C:

  A stack memóriában primitíveket és egy metódushoz tartozó heap-en található objektumok referenciáit tároljuk. Az objektumok mind a heap-en találhatóak.

- [ ] D:

  A heap maximális mérete 1TB (1 Terrabyte vagyis 1024 Gigabyte). Ennél nagyobb heap-et java alkalamzás a memória címezes korlátozásai miatt nem tud használni.

- [x] E:

  Az összes többi állítás mind hamis.
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

Csak a megadott a helyes válasz.

A: nem helyes. A stackoverflow ettől még bekövetkezhet.

B: a stack-et nem a garbage collector tisztítja

C: a stack-en is jöhet létre objektum escape analysis optimalizáció függvényében.

D: nincs maximális heap méret.

5. feladat 0/4 pont

Milyen esetekben választottunk jól GC algoritmust Java 17 alatt futó alkalmazásunkhoz?

Válaszok

- [x] egy processzorral és 1GB memóriával rendelkező virtuális gépen batch folyamatok futtatása: -XX:+UseSerialGC
  Ez a válasz helyes, de nem jelölted meg.
- [ ] egy processzorral és 1GB memóriával rendelkező virtuális gépen batch folyamatok futtatása: -XX:+UseG1GC
- [ ] konzisztensen alacsony számítási időre kiemelten érzékeny tőzsdei alkalmazás 8 processzorral és 32GB memóriával: -XX:+UseConcMarkSweepGC
- [x] konzisztensen alacsony számítási időre kiemelten érzékeny tőzsdei alkalmazás 8 processzorral és 32GB memóriával: -XX:+UseZGC
  Ez a válasz helyes, de nem jelölted meg.
- [ ] átlagos webalkalmazás 2 processzorral és 4GB memóriával: -XX:+UseConcMarkSweepGC
- [ ] átlagos webalkalmazás 2 processzorral és 4GB memóriával: -XX:+UseEpsilonGC

Magyarázat

Egy processzoron batch műveletekhez a SerialGC a leghatékonyabb.

A ConcMarkSweepGC Java 17 alatt már nem működik.

Az EpsilonGC nem alkalmas webalkalmazásokhoz, mert hamar meg fog telni a memória.

Alacsony válaszidőhöz és nagy memóriaterülethez a ZGC a leghatékonyabb a beépített szemétgyűjtők közül.

KÉSZÍTETTE cone

Megjelenés

Világos