JAVA programavimo kalba

Java gijos (Threads)

Lygiagretus vykdymas

Kompiuteris vienu metu gali atlikti kelias užduotis, t.y. vykdyti vienu metu kelias programas. Tada sakoma, kad tokios programos vykdomos lygiagrečiai (concurrent).

- Lygiagrečiame programavime išskiriami du baziniai vykdymo vienetai - procesai (processes) ir gijos (threads)
- Kompiuteryje paprastai vienu metu vykdomi daug procesų ir daug gijų

Lygiagretus vykdymas

Procesas (**process**) - tai vykdoma atskira programa su savo atskiru kodu, resursais, ir kuri komunikuoja su kitais procesais operacinės sistemos suteikiamomis priemonėmis:

- vamzdžiai (pipes), lizdai (sockets) ir t.t.

Gija (**thread**) - tai toje pačioje programoje veikianti nepriklausoma vykdymo gija. Ir tokių gijų gali būti keletas.

Kiekvienas procesas turi bent vieną giją - kitaip jis neveiktų. Ta gija vadinama “pagrindine” gija (main thread).

Lygiagretus vykdymas JAVA

Java turi programines priemones kurti, vykdyti ir valdyti gijas. Gijas galima aprašyti ir startuoti dviem būdais:

1. Sukurti klasės, kuri realizuoja Runnable sąsają, objektą ir jį perduoti naujai sukurtam Thread objektui vykdyti.

2. Sukurti klasės, kuri išplečia Thread klasę, objektą ir jį vykdyti.

Gijos (threads)

Thread klasės metodai:

- currentThread() - statinis metodas, kuris grąžina einamąją giją.
- sleep(long millis), sleep(long millis, int nanos) - statinis metodas nurodytam laikui užmigdo giją.
- interrupt() - nutraukia giją. Jei gija miegojo, tai ji atsibunda ir išmeta InterruptedException. Jei gija nemiegojo tai niekas nevyksta, tik pasikeičia gijos būsena.
- isInterrupted() - statinis metodas patikrina ar einamoji gija yra nutrauktos gijos būsenoje. Gijos būsena po patikrinimo nepasikeičia.
- interrupted() - tas pats tikrinimas kaip ir aukščiau tik po patikrinimo gijos būsena atstatoma į normalia.

Gijos (threads)

Gijas galima jungti į grandinėles, t.y. nurodyti gijai, kad ji turi laukti, kol nesibaigs kita gija ir tik tada pradėti vykdymą.

- joins()
- joins(long millis)
- joins(long millis, int nanos) - metodas sustabdo einamąją giją ir laukia tol kol gija, kurios joins() metodas iškviestas, baigsis. Jei nurodytas laikas, tai gija stabdoma ne ilgiau kaip nurodyta.

Galimi Gijų konfliktai

```
class Counter {
        private int c = 0;
        public void increment() { c++; }
        public void decrement() { c—; }
        public int value() { return c; }
}
```

- Thread A: Paima c reikšmę (0)
- Thread B: Paima c reikšmę (0)
- Thread A: Padidina (1)
- Thread B: Sumažina (-1)
- Thread A: Išsaugoja, c = 1
- Thread B: Išsaugoja, c = -1
- Galutinė c reikšmė -1, nors turėtų būti 0.

Galimi Gijų konfliktai - sprendimas

```
class SynchronizedCounter {
        private int c = 0;
        public synchronized void increment() { c++;}
        public synchronized void decrement() { c—;}
        public synchronized int value() { return c; }
}
```

Sinchronizacija

Ne visada naudinga pažymėti visą metodą kaip sinchronizuotą (synchronized) jei užtenka sinchronizuoti tik tam tikrus veiksmus.

- Reikia naudoti synchronized(*objektas*) { … } sakinį
- Kaip sinchronizavimo objektas gali būti nurodytas, bet koks objektas, kuris veikia kaip sinchronizavimo raktas, užrakinantis arba atrakinantis bloką vykdymui gijose.
- Jei sinchronizuojami keli susiję (!) blokai, tai kaip raktas turi būtų nurodytas tas pats objektas.

Sinchronizuotos kolekcijos

Visos kolekcijos (List, Set, SortedSet, Map, SortedMap) yra nepritaikytos dirbti daugiagijame (multithread) režime. Egzistuoja kolekcijų klasės apvalkalai (wrapper) skirtos valdyti kolekcijas daugiagijame režime. Tokios kolekcijos sukuriamos per taip vadinamus metodus-fabrikus (t.y. tokiemetodai, kurie sukuria ir grąžina naują objektą):

- List<Type> list = Collections.synchronizedList(new ArrayList<>());
- Set<Type> set = Collections.synchronizedSet(new HashSet<>());
- SortedSet<Type> set = Collections.synchronizedSortedSet(new TreeSet<>());
- Map<Key, Value> map = Collections.synchronizedMap(new HashMap<>());
- SortedMap<Key, Value> map = Collections.synchronizedSortedMap(new TreeMap<>());
- Collection<Type> collection = Collections.synchronizedCollection(new ArrayList<>());

Sinchronizacijos problemos

Reikia labai atsargiai naudoti - nes gali iššaukti kelias problemas:

- “Mirtinas užrakinimas” (**deadlock**) - tai pagrindinė problema, kai dvi gijos stovi ir laukia viena kitos užrakintų resursų.
- “Badavimas” (**starvation**) - reta situacija, kai viena gija pastoviai laukia ilgai trunkančios resurso, nes kita gija taip dažnai jį naudoja, kad jis beveik visą laiką pasidaro užimtas.
- “Gyvas užrakinimas” (**livelock**) - reta situacija, kai vienos gijos veikimas priklauso nuo kitos ir abi gijos bando veikti bet negali progresuoti (analogas realiame pasaulyje kai du žmonės bando prasilenkti siaurame koridoriuje pastoviai bandydami vienas kitą užleisti - žmonės juda, bet į priekį nepraeina).

Gijos

Jei vienos gijos veikimą reikia sinchronizuoti su kitos atsižvelgiant į kažkokių kintamųjų reikšmes.

- Object.wait() - sustabdo einamosios gijos veikimą iki tol, kol kitoje gijoje nebus iškviestas to pačio objekto notify() arba notifyAll().
- wait(), notify() ir notifyAll() galima kviesti tik tam objektui, kuris jau yra einamosios gijos užrakintas (!!!)

Uždaviniai

1. Sukurkite programą kur keletas objektų (tarkime kokie 5) vienu metu sugeneruoja po šimtą atsitiktinių skaičių iš intervalo 1..100 (imtinai). Išveskite tuos skaičius didėjimo tvarka kartu su pasikartojimo skaičiumi, pvz.: 1 (5), 3 (1), … - tai reiškia, kad skaičius 1 buvo sugeneruotas 5 kartus, skaičius 2 – visai nebuvo, 3 - vieną kartą ir t.t.