Konkurentno programiranje

Sadržaj

Uvod

- Kod klasičnog pristupa programiranju instrukcije se izvršavaju redom jedna za drugom, odnosno sekvencijalno
- Redosled izvršavanja je uvek isti i samim ti predvidiv
- U realnom računarskom sistemu više programa se izvršava istovremeno i svaki zahteva vreme procesora da bi se izvršio

Uvod

- Kada bi svaki program bio sekvencijalan, računar ne bi mogao efikasno da iskorištava resurse
- Pogotovo treba uzeti u obzir činjenicu da današnji računari poseduju višejezgarne procesore čiji potencijal treba potpuno iskoristiti
- Npr. program za razmenu poruka koji blokira unos dok se ne pošalje poruka

Uvod

- Primer sekvencijalnog programa (operacija posaljiPoruku dugo traje)

```
public class InstantMessenger {
    public static void main(String [] args){
        String poruka;
        do {
            poruka = ucitajPoruku();
            posaljiPoruku(poruka);
        } while(poruka != null);
    }
}
```

Uvod

- Na scenu stupa konkurentno programiranje
- Izvršavanje programa se deli na niti (eng. Thread): paralelne tokove izvršavanja programa
- Program može svaku dugotrajnu operaciju izvršavati u posebnoj niti, što znatno povećava stepen iskorišćavanja resursa računara
- Npr. program za razmenu poruka može slanje poruke obaviti u posebnoj niti, bez uticaja na dalje izvršavanje programa

Niti u Javi (klasa Thread)

- Java program je po definiciji konkurentan jer se glavni program izvršava u glavnoj niti (eng. main thread)
- Programske niti u Javi su realizovane pomoću Thread ili interfejsa Runnable
- Recept za pravljenje niti uz pomoć klase Thread:
  - Napisati klasu koja nasleđuje klasu Thread
  - Redefinisati metodu run, u njoj napisati programski kod niti
  - Nit se pokreće instanciranjem ove klase I pozivanjem njene metode start
- Kod unutar run metode se izvršava sekvencijalno, ali odvojeno od glavnog toka programa

Klasa Thread

```
public class MojThread extends Thread {
    public void run() {
        // programski kod niti je ovde
    }
}
...
// pokretanje niti
MojThread mt = new MojThread();
mt.start();
```

Niti u Javi (interfejs Runnable)

- Drugi način za kreiranje niti je uz pomoć interfejsa Runnable
- Recept za pravljenje niti uz pomoć interfejsa Runnable:
  - Napisati klasu koja implementira interfejs Runnable
  - Redefinisati metodu run, u njoj napisati programski kod niti
  - Instancirati objekat ove klase
  - Instancirati objekat klase Thread i proslediti objekat naše klase u konstruktoru
  - Pozvati metodu start objekta klase Thread

Runnable interfejs

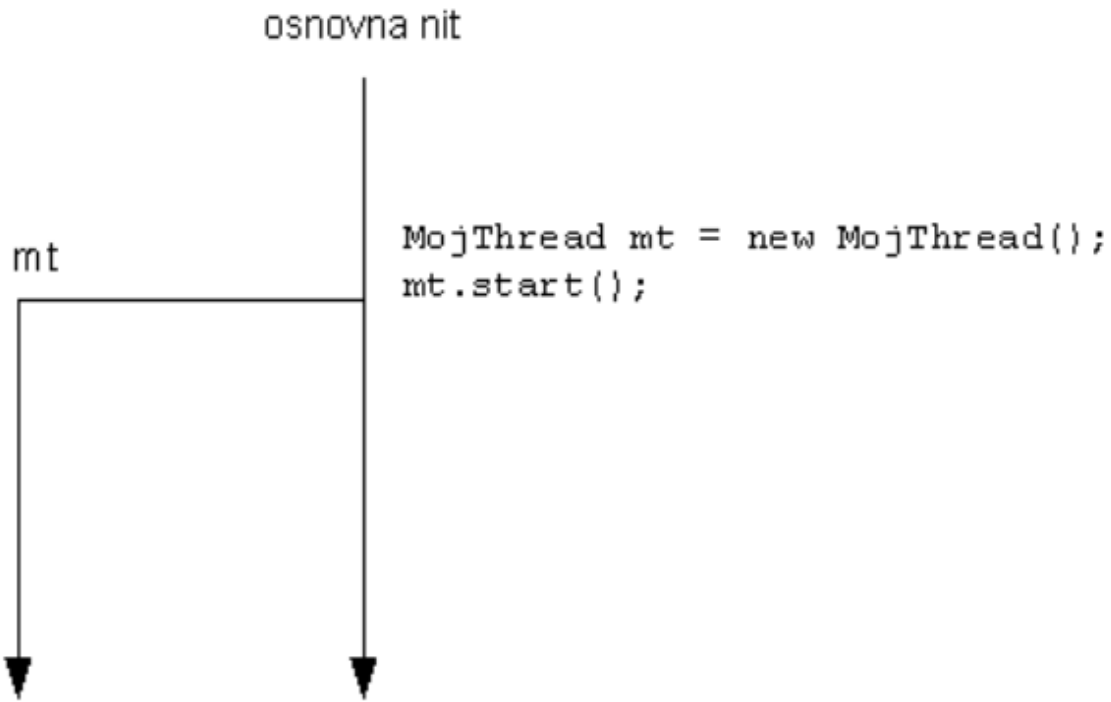
```
public class MojThread implements Runnable {
    public void run() {
        // programski kod niti je ovde
    }
}
...
// pokretanje niti
MojThread mt = new MojThread();
Thread t = new Thread(mt);
t.start();
```

Primer

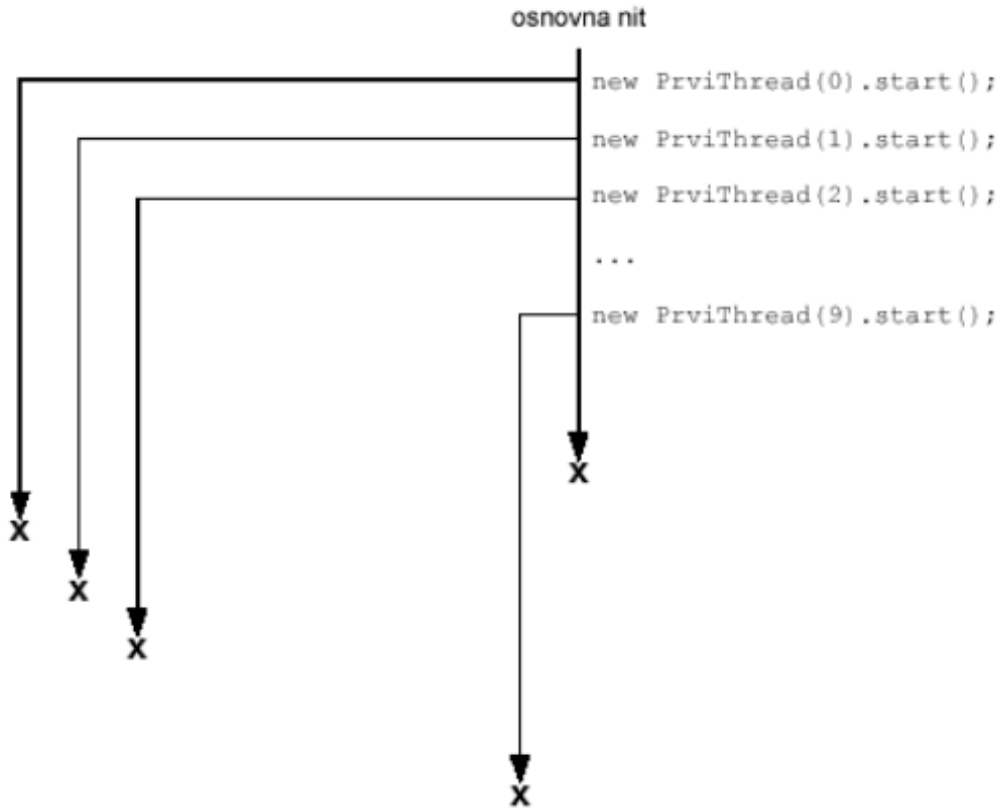
- Program za razmenu poruka, sada sa nitima:

```
public class PorukaNit extends Thread {
    public void run(){
        posaljiPoruku(poruka);
    }
}
...
public class InstantMessenger {
    public static void main(String [] args){
        String poruka;
        do {
            poruka = ucitajPoruku();
            PorukaNit nit = new PorukaNit();
            nit.start();
        } while(poruka != null);
    }
}
```

Izvršavanje glavne niti i jedne dodatne niti



Izvršavanje više niti



Daemon niti

- Do sada je bilo reči o regularnim, odnosno non-daemon nitima
- Java virtuelna mašina (JVM) će čekati da se svaka non-daemon nit završi, čak i ako se glavna nit završila
- Daemon su niti takve da JVM ne čeka kraj njihovog izvršavanja
- Daemon niti obično izvršavaju poslove koje je potrebno izvršavati u pozadini za celokupno vreme izvršavanja programa
- Jedan primer ovakve niti je server za prijem poruka, koji će se izvršavati u pozadini i stalno čitati pristigle poruke
- Recept za kreiranje Daemon niti:
  - Pozvati setDaemon(true) objekta klase koja nasleđuje klasu Thread ili implementira interfejs Runnable

sleep(n)

- Metoda sleep(n) zaustavlja izvršavanje niti iz koje je pozvana na n milisekunde
- Statička metoda Thread.currentThread() vraća referencu na programsku nit iz koje je pozvana
  - Ovo se obično koristi da se dobavi referenca na glavnu nit iz main() metode
- Napomena: metoda sleep se mora pozvati unutar try-catch bloka:

```
try {
        this.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

join() metoda

- Nekada je potrebno izvršiti više paralelnih poslova da bi se nastavilo dalje
- Potreban je mehanizam koji će omogućiti "čekanje" na kraj izvršavanja ostalih niti iz niti koja se ranije završila
- join() metodom se čeka na završavanje niti
- Ako postoji više niti čije se završavanje čeka, potrebno je pozvati join() za svaku od njih
- Napomena: posle poziva join() metode za neku nit, nemoguce je ponovo je pokrenuti: mora ponovo da se instancira sa new Thread()

join() metoda

```
thread2.start();
// sada glavna nit i thread 2 rade paralelno
thread2.join(); // sada cekamo da thread2 zavrsi
System.out.println("Sacekali");
```

yield() metoda

- Nit koja je pozvala metodu yield() prebacuje kontrolu na neku drugu
- Ova metoda privremeno se suspenduje nit koja je pozvala ovu metodu i izvršava se neka druga
- Nemoguće je odabrati koja nit će se sledeća izvršiti (ovo odlučuje operativni sistem)

Primer: auto trka

- Primer implementira trku automobila u kojem svaka nit predstavlja jedan automobil
- Automobili voze neko vreme (između 0 i 2 sekunde), ulaze u pit stop (1 sekunda), i na kraju nastavljaju da voze još neko vreme (između 0 i 2 sekunde)