Predavanje – Projektovanje softvera - Niti

Sadržaj:

1.Niti

- Edsger Dijkstra je rekao da se *"konkurentnost dešava kada postoje dva ili više izvršnih tokova (procesa) koji su sposobni da se izvršavaju simultano (istovremeno)"*.
- Deljeni resursi
- Međusobno isključenje (Mutual exclusion)
- Mrtvo zaključavanje (Deadlock) i Gladovanje (Starvation)
- Ukoliko više procesa međusobno sarađuju u izvršenju nekog zadatka javlja se problem njihove međusobne komunikacije i razmene podataka jer svaki proces zauzima poseban memorijski prostor. Taj problem je rešen pojavom **niti (threads)** koje dele isti memorijski prostor.
- To znači da jedan program (proces) može da obavlja više niti istovremeno (konkurentno).
- ***Nit predstavlja deo programa koji može istovremeno da se izvršava sa drugim nitima istog programa.***

Glavna programska nit

Kada program u Javi počne da se izvršava, on automatski kreira i izvršava jednu nit koja se zove glavna programska nit.

// Primer NT1: Napisati program koji ce da ukaze na glavnu nit koju treba uspavati 5 sekundi.

```
class NT1
{ public static void main(String args[])
  {   Thread gnit = Thread.currentThread();
      System.out.println("Glavna nit:" + gnit + '\n' + "Pauza od 5 sekundi");
      try {    Thread.sleep(5000); // 5 sekundi pauze
          } catch (InterruptedException e) { System.out.println("Prekid niti");}
      gnit.setName("Glavna"); // Promena naziva niti
      System.out.println("Glavna nit:" + gnit + '\n' + "Naziv glavne niti:" + gnit.getName());
  }
}
```

```
// Rezultat:
// Glavna nit:Thread[main,5,main]
// Pauza od 5 sekundi.
// Glavna nit:Thread[Glavna,5,main]
// Naziv glavne niti: Glavna
```

Pravljenje niti

U Javi se nit može napraviti na 2 načina:

- Realizacijom interfejsa Runnable
- Proširenjem klase Thread

Pravljenje niti realizacijom klase Runnable

Kada se nit pravi realizacijom interfejsa Runnable, tada je jedino potrebno se implementira (realizuje) metoda run() interfejsa Runnable.

```
// Primer NT2: Napraviti nit pomocu interfejsa Runnable.

class NT2 implements Runnable
{  NT2()
   { nit = new Thread(this,"Nova nit"); //nova nit ce pozvati run metodu od this objekta
     nit.start();
   }
 public void run()
 { System.out.println("Nit:" + nit); }

 public static void main(String args[])
      { NT2 nn = new NT2();
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);
      }
  Thread nit;
}

// Rezultat:
// Nit:Thread[Glavna nit,5,main]
// Nit:Thread[Nova nit,5,main]
```

Pravljenje niti proširenjem klase Thread

Kada se nit pravi proširenjem klase Thread, tada je potrebno se prekrije metoda run() klase Thread. Klasa Thread pored metode run() sadrži i druge metode koje mogu da se prekriju, za razliku od interfejsa Runnable koji ima samo metodu run().

```
// Primer NT3: Napraviti nit pomocu klase Thread.

class NT3 extends Thread
{
 NT3()
 { super("Nova nit");
   start();
 }

 public void run()
 { System.out.println("Nit:" + currentThread()); }

 public static void main(String args[])
      { NT3 nn1 = new NT3();
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);

      }
 }
// Rezultat:
// Nit: Thread[Glavna nit,5,main]
// Nit: Thread[Nova nit,5,main]
```

Pravljenje više niti

U jednom Java programu može da postoji više niti.

```
// Primer NT6: Napraviti u jednom programu vise niti.

class NT6 extends Thread
{ NT6(String nn)
 { super(nn);
   start();
 }
 public void run() { System.out.println("Nit:" + currentThread()); }

 public static void main(String args[])
      { NT6 nn1 = new NT6("Nova nit1");
        NT6 nn2 = new NT6("Nova nit2");
        NT6 nn3 = new NT6("Nova nit3");
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");
        System.out.println("Nit:" + gnit);
      }
 }

// Rezultat:
// Nit: Thread[Glavna nit,5,main]
// Nit: Thread[Nova nit1,5,main]
// Nit: Thread[Nova nit2,5,main]
// Nit: Thread[Nova nit3,5,main]
```

Stanja niti

Nit može biti u jednom od 4 stanja:

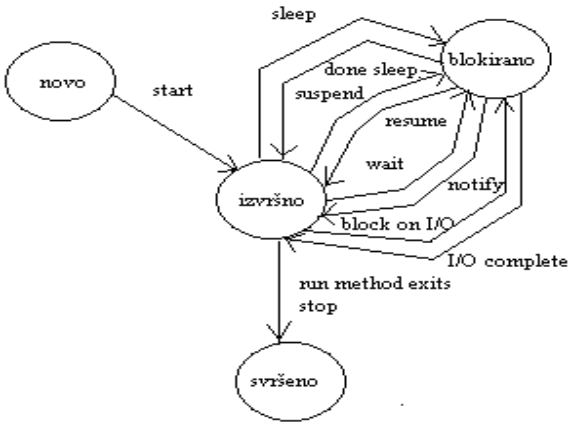
- novo (new)
- izvršno (runnable)
- blokirano (blocked)
- svršeno (dead)

```
class NT3 implements Runnable   // NapraviNit.java
{
 NT3()
 { nit = new Thread(this,"Nova nit"); // Nit je u stanju ”novo”
   nit.start(); // Nit je u stanju ”izvrsno”
 }

 public void run()
 { System.out.println("Nit:" + nit); } // Nit je u stanju “izvrsava”
// Nakon izvrsenja tela metode run() nit prelazi u stanje ”svrseno”

 public static void main(String args[])
      { NT3 nn = new NT3();
        Thread gnit = Thread.currentThread();
        gnit.setName(“Glavna nit”);
        System.out.println(“Nit:” + gnit);
      }

  Thread nit;
}
// Rezultat:
// Nit: Thread[Glavna nit,5,main]
// Nit: Thread[Nova nit,5,main]
```

Ulazak niti u blokirano stanje

Nit prelazi u blokirano stanje, kada se desi jedna od sledećih akcija:

- Nit poziva *sleep()* metodu. Nit ostaje ”*uspavana*” zadati broj milisekunda.
- Nit poziva operaciju, koja je blokirala neki od ulazno/izlaznih uređaja. Takva operacija se neće nastaviti, sve dok se ne oslobodi zauzeti ulazno/izlazni uređaj.
- Niz poziva *wait()* metodu.
- Nit pokušava da zaključa objekat koji je već zaključan od strane druge niti.
- Nit poziva suspend *metodu()*. Ova metoda je zastarela i više se ne koristi.



Izlazak niti iz blokiranog stanja

Nit može da pređe iz ”blokiranog” u ”izvršno” stanje, kada se desi jedna od sledećih akcija:

- Nit se ”probudila” nakon što je prošlo vreme ”uspavanosti ”, zadato metodom sleep().
- Operacija, koja se pozvana u toku izvršenja niti, koja je blokirala ulazno/izlazni uređaj je završena.
- Ako je prva nit pozvala wait() metodu, druga nit mora pozvati notify() ili notifyAll() metodu kako bi se deblokirala prva nit.
- Kada blokirana nit, koja čeka na zaključan objekat, dobije kontrolu nad njim, nakon što je neka nit prestala da drži taj objekat zaključan.
- Ako je nit bila blokirana metodom suspend(), ona se može deblokirati jedino pozivom metode resume(). Metoda resume() je takođe zastarela kao i metoda suspend()

Važna napomena:

Blokirana nit se može deblokirati jedino inverznom operacijom od operacije koja ju je blokirala (sleep – done sleeping, suspend – resume, wait – notify, block on I/O - I/O complete)

Ako se pokuša deblokirati nit sa neodgovarajućom inverznom operacijom javiće se *IllegalThreadStateException* izuzetak.

Uništenje niti

Nit prelazi u stanje ”*svršeno*” u sledeće 2 situacije:

- run() metoda je prestala da se izvršava prirodnim putem.
- run() metoda je prestala da se izvršava jer se u toku njenog izvršenja desio neuhvaćen izuzetak.

Nit se takođe može zaustaviti ako se pozove metoda stop(). Međutim ova metoda je zastarela i ne koristi se jer dovodi do nestabilnog ponašanja programa.

.

Prekid niti

- Nit se završava kada se telo run() metode izvrši. Pošto je stop() metoda prevaziđena, koristi se logička kontrola za nastavak izvršenja niti:

```
public void run
{  while(signal)   { izvrsenje niti…   }
}
```

- Nit prestaje da se izvršava kada signal dobije *false* vrednost..

```
public void promeni()
{signal = false;}
```

- Međutim u slučaju kada je nit u blokiranom stanju, prekid izvršenja niti ne može se ostvariti na predhodni način.
- interrupt() metoda poziva blokiranu nit generisanjem izuzetka InterruptedException.
- Blokada može prestati ako je ista nastala na osnovu sleep() ili wait() metode.
- Prekida se blokada, ne prekida se izvršenje niti.
- Ako se želi prekid niti:

```
public void run
{
  try {…
        while(signal)
         { izvrsenje niti…   }
      } catch(InterruptedException e)
          { ... }
…
// izlaz iz run() metode i prekid izvršenja niti.
}
```

Metode interrupt(), interrupted() i isInterrupted()

Metoda interrupt() status niti postavlja na true.
U slučaju kada interrupt() metoda pozove nit koja nije blokirana, tada se neće desiti izuzetak InterruptedException. Zbog toga se kod logičke kontrole izvršenja niti poziva metoda interrupted(), koja proverava da li je tekuća nit "pretrpela" interrupt() metodu (da li je status niti true).
Ukoliko je status niti true treba da se prekine izvršenje niti:

```
while( !interrupted() && signal)
   { izvrsenje niti…
   }
```

ili

Metoda interrupted() ima spoljni efekat (side effect), jer stanje niti postavlja na false.
Pored navedenih postoji i metoda isInterrupted() koja vraća stanje prekida niti ali nema spoljni efekat:

```
while( !isInterrupted() && signal)
   { izvrsenje niti…
   }
```

Primeri:

Primer NT4

Primer NT4 *: Pokazati kako se prekida "uspavanost" niti pomocu metode interrupt().*

```
class NT4 extends Thread
{  NT4() { start(); }

 public void run()
 {   try {     sleep(5000);     } catch(InterruptedException e) {}
            System.out.println("Nova nit probudjena!");  }

 public static void main(String args[]) throws Exception
      {  NT4 nn = new NT4();
        Thread gnit = Thread.currentThread();
        gnit.sleep(2000);
        System.out.println("Glavna nit probudjena!");
        nn.interrupt(); // Kada se iskljuci ova naredba nova nit nn je uspavana
        // 5 sekundi, inace se odmah prekida "uspavanost" niti nn.
      }
}

// Rezultat:
// Glavna nit probudjena
// Nova nit probudjena
```

Sledeci primer:

Primer NT5 (TextPad):

```
// Primer NT5: Pokazati kako se prekida izvrsenje niti u 3 situacije:
// a) Ukoliko se nit izvrsava u while petlji i nit nije uspavana.
// b) Ukoliko se nit izvrsava u while petlji i  nit je uspavana.
// c) Ukoliko se nit izvrsava u while petlji i  nit nije uspavana a zelimo je
// prekinuti preko interrupt() metode.

class NT5 extends Thread
{  NT5(int opcija1)
   { signal = true;
     opcija = opcija1;
    start();
  }

 public void run()
 {  switch(opcija)
     { case 0:  ObicanPrekidNiti(); break;
       case 1:  InterruptKojiNePrekidaIzvrsenjeNiti();break;
       case 2:  InterruptKojiPrekidaIzvrsenjeNiti();break;
       case 3:  InterruptKojiPrekidaNeuspavanuNit();break;
   }
 }

// Iz glavnog programa promenice se status signal promenljive sto ce dovesti do prekida
// niti.
public void ObicanPrekidNiti()
 {  while(signal) { }
  System.out.println("Nova nit 0 probudjena!");
 }
// Prekid "uspavanosti" niti vratice kontrolu izvrsenja niti u while petlju. Ovde je
// problem u tome sto se try/catch blok nalazi unutar while petlje.
public void InterruptKojiNePrekidaIzvrsenjeNiti()
 { while(signal)
      { try {sleep(5000);}
        catch(InterruptedException e) {}
      }
   System.out.println("Nova nit 1 probudjena!"); }

// Prekid "uspavanosti" niti prekinuce izvrsenje.niti, jer je try/catch blok izvan
// while petlje a ne unutar nje.
public void InterruptKojiPrekidaIzvrsenjeNiti()
 {  try {   while(signal)
              { sleep(5000); }
         } catch(InterruptedException e) {}
   System.out.println("Nova nit 2 probudjena!");
  }

// Ukoliko se desi da interrupt metoda pozove neuspavanu nit potrebno je izvrsiti
// dopunsku kontrolu while petlje pomocu interrupted() metode koja vraca true ukoliko
// je za tekucu nit pozvana interrupt() metoda.
```

```
public void InterruptKojiPrekidaNeuspavanuNit()
 {
   // ** while(!isInterrupted() && signal) { } // ne menja interrupt status niti
   while(!interrupted() && signal) { } // menja interrupt status niti na false
   System.out.println("Nova nit 3 probudjena!");
   System.out.println("Status prekida : " + isInterrupted());
   // Status prekida bi bio true da se izvrsila naredba **
  }

public void prekid()
  { signal = false;}

 public static void main(String args[]) throws Exception
      {  NT5 nn0 = new NT5(0);
         NT5 nn1 = new NT5(1);
         NT5 nn2 = new NT5(2);
         NT5 nn3 = new NT5(3);
        Thread gnit = Thread.currentThread();
        gnit.sleep(2000);
        System.out.println("Glavna nit probudjena!");
        nn0.prekid();   nn1.interrupt();   nn2.interrupt();    nn3.interrupt();
      }
boolean signal;
int opcija;
}

// Rezultat:
// Glavna nit probudjena
// Nova nit 2 probudjena
// Nova nit 0 probudjena
// Nova nit 3 probudjena
// Status prekida: false
```

Zadatak NTZ1: Napisati 2 niti tako da jedna od niti u toku svog izvrsavanja prekine izvrsavanje druge niti.
Zadatak NTZ2: Napisati 2 niti tako da jedna od niti u toku svog izvrsavanja “uspava” drugu nit.

Korišćenje metoda isAlive() i join()

Ukoliko se želi utvrditi, da li se nit još izvršava, koristi se metoda isAlive() klase Thread. Navedena metoda vraća true, ukoliko se nit još izvršava, odnosno false, ukoliko je nit izvršena.
Ukoliko se želi sačekati sa izvršenjem nekog dela programa dok se neka od niti ne izvrši koristi se metoda join() klase Thread.

Primer:

Primer NT7

```
// Primer NT7: Pomocu metoda isAlive() proverti da li se nit jos izvrsava.
// Pomocu metode join() pokazati kako glavna nit ceka na izvrsenje drugih niti
// pre nego sto zavrsi program.

class NT7 extends Thread
{
 NT7(String nn)
 { super(nn);   start();
 }

 public void run()
 { System.out.println("Nit:" + currentThread()); }

 public static void main(String args[])
      { NT7 nn1 = new NT7("Nova nit1");
        NT7 nn2 = new NT7("Nova nit2");
        System.out.println("Nova nit1 se izvrsava:" + nn1.isAlive());
        System.out.println("Nova nit2 se izvrsava:" + nn2.isAlive());
        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");

        try{   System.out.println("Pre kontrole nit1");
        nn1.join(); System.out.println("Nova nit1 se izvrsava:" + nn1.isAlive());
        System.out.println("Pre kontrole nit2");
        nn2.join(); System.out.println("Nova nit2 se izvrsava:" + nn2.isAlive());
                  }
        catch(InterruptedException e) { System.out.println("Prekid glavne niti" );}
        System.out.println("Nit:" + gnit);
      }
 }
`
// Rezultat:
// Nova nit1 se izvrsava:true
// Nova nit2 se izvrsava:true
// Pre kontrole nit1
// Nit:Thread[Nova nit1,5,main]
// Nova nit1 se izvrsava:false
// Pre kontrole nit2
// Nit:Thread[Nova nit2,5,main]
// Nova nit2 se izvrsava:false
// Nit:Thread[Glavna nit,5,main]
```

Zadatak NTZ3: Napisati 2 niti tako da druga nit pocne da se izvrsava tek kada se zavrsi prva nit. U radu koristiti metode isAlive() i join().

Prioritet izvršavanja niti

- U Javi svaka nit ima svoj prioritet. Podrazumevano nit nasleđuje prioritet roditeljske niti. Prioritet niti može da se promeni sa metodom *setPriority()*. Prioritet može da uzme jednu od vrednosti iz opsega od 1 (MIN_PRIORITY) do 10 (MAX_PRIORITY). Podrazumevani prioritet svake niti, osim ako se drugačije ne zada je 5 (NORM_PRIORITY).

- Nit nekog nivoa prioriteta se izvršava dok:
  1. se ne pozove metoda yield() koja prekida izvršenje tekuće niti, dopuštajući da druga nit, istog ili višeg prioriteta počne da se izvršava.
  2. ne pređe u stanje "blokiran" ili "svršen".
  3. nit višeg prioriteta ne postane izvršna (zato što se "probudila" ili je U/I operacija kompletirana ili je neko pozvao notify() metodu.

Primer:

Primer NT8

```
// Primer NT8: Napisati program koji ce da procentualno pokaze zauzetost procesora
// od strane vise niti, u zavisnosti od prioriteta niti.

import java.text.*;

class NT8 extends Thread
{  NT8(String nn,int Prioritet)
    { super(nn);
      setPriority(Prioritet);
     start();
   }

 public void run()
 { while (signal) { brojac++;}
 }

 public void prekini() { signal = false;}
 public static void main(String args[])
      { long SumaKoraka;
        NumberFormat nf = NumberFormat.getNumberInstance();

        NT8 nn1 = new NT8("Nova nit1",5); // Najvisi prioritet
        NT8 nn2 = new NT8("Nova nit2",4); // Najnizi prioritet

        Thread gnit = Thread.currentThread();
        gnit.setName("Glavna nit");

       try{ gnit.sleep(6000); // 6 sekundi pauza
          }  catch(InterruptedException e) { System.out.println("Prekid glavne niti" );}

        nn1.prekini();
        nn2.prekini();

        try{
        nn1.join(); // Zaustavlja se izvrsenje glavne niti dok se ne zavrsni nit nn1.
        nn2.join(); // Zaustavlja se izvrsenje glavne niti dok se ne zavrsni nit nn2
           }  catch(InterruptedException e) { System.out.println("Prekid glavne niti" );}

        SumaKoraka = nn1.brojac + nn2.brojac;
        double p1 = (double) nn1.brojac / SumaKoraka * 100.00;
        Double proc1 = new Double(p1);

        // Konverzija se vrsi u double jer su nn1.brojac i SumaKoraka celi brojevi.
        double p2 = (double) nn2.brojac / SumaKoraka * 100.00;
        Double proc2 = new Double(p2);

        System.out.println("Nova nit 1 - broj koraka:" +nn1.brojac
        + " Procenat:" + nf.format(proc1));

        System.out.println("Nova nit 2 - broj koraka:" + nn2.brojac
        + " Procenat:" + nf.format(proc2));
       }
```

```
   private boolean signal = true;
   long brojac = 0;
 }

// Rezultat:
// a)
// Nit 1 ce zauzeti priblizno 98% procesorskog vremena ako ima priotitet 10
// Nit 2 ce zauzeti priblizno 2% procesorskog vremena  ako ima priotitet 1
// b)
// Nit 1 ce zauzeti priblizno 50% procesorskog vremena ako ima priotitet 5
`           // Nit 2 ce zauzeti priblizno 50% procesorskog vremena  ako ima priotitet 5
// c)
// Nit 1 ce zauzeti priblizno 96% procesorskog vremena ako ima priotitet 5
// Nit 2 ce zauzeti priblizno 4% procesorskog vremena  ako ima priotitet 4
```

Zadatak NTZ4: Napisati 2 niti sa različitim prioritetima izvršavanja. U toku rada izmenite prioritete između niti. Pokazite kako je troseno procesorsko vreme od strane niti pre i posle promene prioriteta.

Sebične (selfish) niti

Niti bi trebale periodično da pozivaju yield() ili sleep() metodu kako bi drugim nitima pružile šansu da se izvrše. Na taj način nit ukida mogućnost da iz nekog razloga ima monopol nad sistemom. One niti koje ne daju drugim nitima mogućnost da se izvrše, nazivaju se “sebične” niti.

Grupe niti

Grupa niti se kreira na sledeći način:

String imeGrupe = "novaGrupa";
ThreadGroup g = new ThreadGroup(imeGrupe);

Nit, koja će se zvati *"novaNit1"* se dodaje do grupe *g* na sledeći način:
Thread t = new Thread(g,"novaNit1");

Ukoliko se želi proveriti da li je neka od niti još u stanju "izvršno", koristi se metoda *activeCount()* na sledeći način:

```
if (g.activeCount() = = 0)
  { // sve niti u grupi niti su zastavljene.
  }
```

Ukoliko se žele prekinuti sve niti u grupi niti poziva se interrupt() metoda nad grupom niti:
g.interrupt();

Grupa niti može da ima svoju podgrupu niti. Metode *activeCount()* i *interrupt()* se odnose na tekuću grupu niti i sve njene podgrupe niti. To znači, npr. da kada se prekida izvršenje jedne grupe niti, sve njene podrgrupe niti takođe prekidaju izvršenje.

Primer:

` Primer NT9

```
// Primer NT9: Pokazati kako se kreira i prekida grupa niti.

class NT9 extends Thread
{
 NT9(NT9G g,String imeNiti)
 {  super(g.tg,imeNiti);    start();
 }
 public void run()
 { while(!interrupted() && true) {}
   System.out.println("Prekinuta nit");
 }
 public static void main(String args[]) throws Exception
      { String imeGrupe = "novaGrupa";
        NT9G g = new NT9G(imeGrupe);
        NT9 t1  = new NT9(g,"novaNit1");
        NT9 t2  = new NT9(g,"novaNit2");
        NT9 t3  = new NT9(g,"novaNit3");
        g.prekini();
      }
}
class NT9G extends Thread
{  NT9G(String imegrupe)
   { tg = new ThreadGroup(imegrupe);
     start();
   }

 public void run()
  { while(true)
    {    if (tg.activeCount() == 0)
           { System.out.println("Sve niti u grupi su prekinute! " + tg.activeCount());   break;    }
           }
  }
  void prekini() { tg.interrupt();}
 ThreadGroup tg;
}
```

Zadatak NTZ5: Napisati grupu niti koja prati stanje nekog objekta. Kada objekat dobije zadatu vrednost prekinuti izvrsenje grupe niti.

Sihronizacija

- Ukoliko se javi potreba da dve ili više niti dele zajednički resurs, pri čemu niti ne mogu istovremeno da koriste zajednički resurs, mora se obezbediti mehanizam koje će omogućiti isključivi (ekskluzivni) pristup jedne niti do zajedničkog resursa. Tek nakon završetka obrade zajedničkog resursa od strane jedne niti, moguće je da druga nit, takođe isključivo, pristupi do njega.

- Monitor obezbeđuje mehanizam za isključivi pristup niti do zajedničkog resursa. Kada neka nit X uđe u monitor, nijedna druga nit ne može u njega ući, sve dok nit X ne izađe iz njega. Postupak kojim monitor obezbeđuje navedeni mehanizam naziva se sinhronizacija.

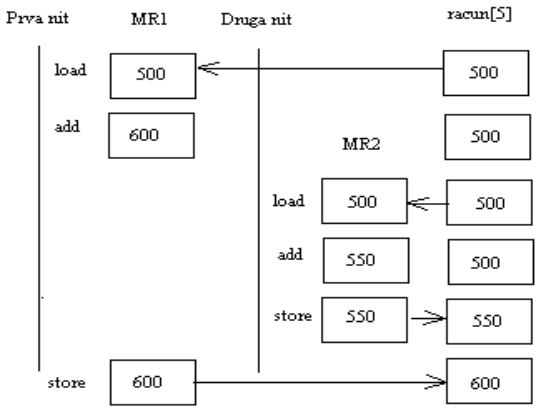
- Potreba istovremenog pristupa do deljenog objekta dovodi do tzv. race condition[1] situacija.

[1] Kada na trkama 2 takmičara, skoro u isto vreme prolaze kroz cilj, javlja se problem preciznog određivanja ko je prvi prošao kroz cilj.

Komunikacija niti bez sinhronizacije

Onemogućavanje istovremenog pristupa niti do deljenih objekata se naziva sinhronizacija pristupa. Ukoliko nema sinhronizacije pristupa može se javiti sledeća situacija:
Ukoliko postoji banka sa 10 računa (za svaki račun se pravi posebna nit), između kojih se vrši prenos novca na slučajno izabran način, može se desiti da iznos istog računa istovremeno povećava 2 ili više niti.

Npr. racun[5] se istovremeno povećava pomoću dve niti.
Stanje računa pre promene: *racun[5] = 500;*
Prva nit *: racun[5] +=100;*
Druga nit *: racun[5] +=50;*
Problem se javlja na nivou atomskih operacija naredbi:



Dobijeni iznos je pogrešan. Validna vrednost koju bi racun[5] trebao da ima je **650.**

Zaključavanje objekata

- Kada nit pozove metodu objekta koji treba da se sinhronizuje, objekat postaje zaključan.

- Preciznije rečeno pri pozivu sinhronizovane metode: “Nit nalazi jedini ključ od objekta ispred vrata, ubacuje ključ u bravu od objekta, otključava ga, ulazi u objekat i bravu sa druge strane vrata zaključava. Brava ostaje zaključana sve dok nit ne izađe iz objekta, odnosno dok nit ne izvadi ključ iz brave i stavi ga ispred vrata”.

- Niti koje čekaju na pristup sinhonizovanim metodama deljenog objekta, mogu pristupiti do nesinhronizovanih metoda deljenog objekta.

Korišćenje sinhronizovanih metoda

U Javi svaki objekat ima svoj monitor. Da bi se monitor pokrenuo potrebno je da se objektu pristupi preko jedne od njegovih sinhronizovanih metoda. Ispred metoda koje treba sinhronizovati stavlja se ključna reč **synchronized.** Treba naglasiti da izvršavanje jedne od sinhronizovanih metoda nekog objekta onemogućava pristup do bilo koje druge sinhronizovane metode istog objekta. Tek nakon završetka izvršenja sinhronizovane metode nekog objekta moguće je pozvati neku drugu sinhronizovanu metodu istog objekta. Za nesinhronizovane metode ne važi navedeno ograničenje.

Zadatak:

NT10S1

```
// Primer NT10S1: Omoguciti sinhronizaciju objekta, kome pristupa vise niti.

class Nit extends Thread
{
        Nit(String Naziv, int BrojNiti,Zalihe z)
           { this.Naziv = Naziv;   this.BrojNiti=BrojNiti;this.z = z;  start();}

        public void run()
          { int PreostalaKolicina = z.Uzmi();
          System.out.println("Nit:" + BrojNiti + " Preostala kolicina:" + PreostalaKolicina);
}
        int BrojNiti;
        String Naziv;
        Zalihe z;
}

class Zalihe
{ private int Kolicina;
  Zalihe(){Kolicina = 10;}
   int Uzmi()  // Ukoliko se stavi  synchronized int Uzmi() nece biti problema u rezultatu.
    { Kolicina--;
      try{ Thread.sleep(1000); // 1 sekunda pauze
          }  catch(InterruptedException e) { System.out.println("Prekid niti" );}
      return Kolicina;
   }
}

class NT10S1
{  public static void main(String args[])
     { Zalihe z = new Zalihe();
       Nit nn1 = new Nit("Nit1",1,z);
       Nit nn2 = new Nit("Nit2",2,z);
       Nit nn3 = new Nit("Nit3",3,z);
     }
 }

// Rezultat:
// Nit 1: Preostala kolicina:7
// Nit 2: Preostala kolicina:7
// Nit 3: Preostala kolicina:7
```

Navedeni rezultat pokazuje da nije izvršena sinronizacija objekta z preko metode Uzmi() Da bi se postigao navedeni efekat potrebno je staviti ***synchronized int Uzmi()*** u klasi Zalihe, umesto *int Uzmi():*

```
class Zalihe   // primer: Sinhronizacija.java
{ private int Kolicina;
  Zalihe(){Kolicina = 10;}
synchronized int Uzmi(){…}
}
```

tada će rezultat program biti sledeći:

```
// Rezultat:
// Nit 1: Preostala kolicina:9
// Nit 2: Preostala kolicina:8
// Nit 3: Preostala kolicina:7
```

Zadatak NTZ6: Korišćenjem sinhronizacije onemogućiti da objektu klase Student pristupe istovremeno 2 niti i promene njegove podatke.

Korišćenje sinhronizovanih objekata

U slučaju da preko metoda ne može da se sinhronizuje pristup objektu, moguće je eksplicitno sinhronizovati sam objekat. Opšti oblik eksplicitne sinhronizacije objekta je:

synhronized (objekat) { blok naredbi koje će biti sinhronizovane}

Ukoliko se želi postići eksplicitna sinhronizacija objekta, potrebno je u predhodnom primeru da se metoda run() klase Nit zameni sa:

```
class Nit extends Thread   // primer: NT10S2.java
{…
   public void run()
    { int PreostalaKolicina;
     synchronized (z)
      { PreostalaKolicina = z.Uzmi();}
      System.out.println("Nit:" + BrojNiti + " Preostala kolicina:" +      PreostalaKolicina);
}…}
```

tada će rezultat programa biti sledeći:

```
// Rezultat:
// Nit 1: Preostala kolicina:9
// Nit 2: Preostala kolicina:8
// Nit 3: Preostala kolicina:7
```

Kada se sinhronizacija objekta vrši preko metode, tada se pre izvršenja sinhronizovane metode zaključava objekat. U tom statusu objekat ostaje sve dok se ne izvrši sinhronizovana metoda.
Kada se sinhronizacija objekta vrši eksplicitno, tada se pre izvršenja bloka naredbi koje treba sinhronizovati zaključava objekat. U tom statusu objekat ostaje sve dok se ne izvrši navedeni blok naredbi.