Vježbe 8 - višedretveni programi u programskom jeziku *Java*

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

14. prosinca, 2022.

Kreiranje dretve

Dretve se mogu kreirati i pokretati na **dva načina**: a) **izradom klase koja implementira sučelje** `Runnable`. To sučelje definira jednu metodu `run` koja sadrži kod koji dretva treba izvoditi. Objekt koji implementira sučelje se proslijeđuje kao parametar konstruktora klase `Thread`.

```
1  public class StvaranjeNacinA implements Runnable{
2
3      @Override
4      public void run(){
5          System.out.println("Pozdrav!");
6      }
7
8      public static void main(String args[]){
9          for(int i=0;i<100;i++)
10             new Thread(new StvaranjeNacinA()).start();
11     }
12 }
```

Kreiranje dretve pomoću objekta koji implementira sučelje Runnable.

Kreiranje dretve

b) **Nasljeđivanjem klase** `Thread`. Klasa `Thread` također implementira `Runnable` iako njezina metoda `run` ne radi ništa. Klasa koja nasljeđuje klasu `Thread` može nadjačati metodu `run` i pružati traženu funkcionalnost.

```
1  public class StvaranjeNacinB extends Thread {
2
3      @Override
4      public void run(){
5          System.out.println("Pozdrav!");
6      }
7
8      public static void main(String args[]){
9          for(int i=0;i<100;i++)
10             new StvaranjeNacinB().start();
11     }
12
13 }
```

Kreiranje dretve pomoću klase koja nasljeđuje klasu Thread.

Kreiranje dretve

Prvi pristup koji koristi objekt koji implementira `Runnable` je **generalniji** jer dopušta korištenje objekta proizvoljne klase za stvaranje dretve (kompatibilan je i s pristupima više razine za rad s višedretvenim aplikacijama definiranim u paketu `java.util.concurrent`). Drugi način je možda jednostavniji za korištenje ali zahtjeva da objekt nasljeđuje klasu `Thread`.

Klasa `Thread` sadrži metode za upravljanje dretvama (uključujući statičke metode koje pružaju informaciju o statusu dretve ili utječu na status dretve koja poziva metodu).

Pauziranje izvođenja korištenjem funkcije Sleep

```
1  public class Pauziranje implements Runnable {
2      int id;
3
4      public Pauziranje(int i){ id = i; }
5
6      @Override
7      public void run(){
8        try{
9          for(int i=0;i<10;i++){
10             System.out.println("Pozdrav iz dretve: "+id);
11             Thread.sleep(5); } }//pauziraj na 5ms
12       catch(InterruptedException e){
13           e.printStackTrace(); } }
14
15     public static void main(String args[]){
16         for(int i=0;i<10;i++){
17             System.out.println("Kreiram dretvu: "+(i+1));
18             new Thread(new Pauziranje(i+1)).start(); }
19         System.out.println("Sve dretve pokrenute!"); } }
```

Pauziranje dretve.

Prekidanje izvođenja dretve

```
1  public class PrekidanjeDretve implements Runnable {
2
3      @Override public void run(){
4        try{
5          while(true){
6              System.out.println("Neki zadatak...");
7              Thread.sleep(1000); } }
8        catch(InterruptedException e){
9            System.out.println("Dretva je prekinuta!"); } }
10
11     public static void main(String args[]) throws
       InterruptedException{
12         Thread t = new Thread(new PrekidanjeDretve());
13         t.start();
14
15         Thread.sleep(5000);
16         t.interrupt(); //prekidamo dretvu nakon 5s
17     }
18 }
```

Prekidanje dretve.

Prekid izvođenja dretve

```
1  public class PrekidanjeDretve1 implements Runnable {
2
3   @Override public void run(){
4       while(true){
5          System.out.println("Radim...");
6             if(Thread.interrupted()){ //provjeravamo je li
       dretva prekinuta
7                System.out.println("Oooooo neeeee!");
8                     return;
9                  } } }
10
11      public static void main(String args[]) throws
       InterruptedException{
12          Thread t = new Thread(new PrekidanjeDretve1());
13          t.start();
14          Thread.sleep(10);//prekidamo nakon 10ms
15          t.interrupt();
16      }
17 }
```

Prekidanje dretve.

Čekanje na završetak izvođenja dretvi

Metoda `join` dopušta da jedna dretva čeka završetak izvođenja druge. Ukoliko imamo `Thread t; t.join();`, tada trenutna dretva čeka (pauzira izvršavanje) dok t ne završi izvođenje. `join` je preopterečen na način da dopušta specificiranje vremena čekanja (koristi brojač vremena operacijskog sustava, stoga vrijeme čekanje može ponešto odstupati od zadanog). Kao i `sleep` i `join` vraća `InterruptedException` ukoliko je dretva prekinuta.

Čekanje na završetak izvođenja dretvi

```
1  public class Spajanje implements Runnable {
2
3      @Override
4      public void run(){
5          for(int i=0;i<10;i++)
6              System.out.println("Iteracija u dretvi: "+i);
7      }
8
9      public static void main(String args[])throws
       InterruptedException{
10         System.out.println("Izvodenje glavne dretve!");
11         Thread t = new Thread(new Spajanje());
12         System.out.println("Objekt dretve stvoren");
13         System.out.println("Pokretanje dretve...");
14         t.start();
15         t.join();//cekamo zavrsetak izvodenja dretve t
16         System.out.println("Nastavak nakon spajanja");
17     }
18 }
```

Čekanje na završetak izvođenja dretve.

Sinkronizacija

```
1  public class Brojac {
2       private int c = 0;
3
4      public void increment() { c++; }
5      public void decrement() { c--; }
6      public int getCounter(){ return c; }
7  }
8
9  public class Sinkronizacija implements Runnable{
10     int id;
11     Brojac tmp;
12
13     Sinkronizacija(Brojac r, int _id){
14         tmp = r;  id = _id; }
15       @Override public void run(){
16         if(id == 0){
```

Neispravno sinkronizirani program.

Sinkronizacija

```
            for(int i=0;i<100000;i++){
                tmp.increment(); } }
        else if(id==1){
             for(int i=0;i<100000;i++){
                tmp.decrement(); } } }

public static void main(String args[]) throws
   InterruptedException{
 Brojac b = new Brojac();
 Thread t1 = new Thread(new Sinkronizacija(b,0));
 Thread t2 = new Thread(new Sinkronizacija(b,1));
 t1.start(); t2.start();
 t1.join(); t2.join();
 System.out.println("Vrijednost brojaca nakon 100000
   inkrementa i 100000 dekremenata");
 System.out.println("Brojac: "+b.getCounter());
   }//jedan dobiveni rezultat: 1104 (umjesto 0)
 }
```

Neispravno sinkronizirani program.

Sinkronizacija

Gornji program možemo ispravno sinkronizirati na sljedeći način:

```
1 public class SinkroniziraniBrojac {
2      private int c = 0;
3
4     public synchronized void increment() { c++; }
5     public synchronized void decrement() { c--; }
6     public int value() { return c; }
7     public int getCounter(){ return c; }
8 }
```

Ispravno sinkronizirani program.

Ukoliko svuda zamijenimo `Brojac` sa `SinkroniziraniBrojac`, program svaki puta vrati točnu vrijednost 0.

Sinkronizacija monitorima pridruženim objektima

Sinkronizacija monitorima pridruženim objektima omogućava ekskluzivni pristup objektu i/ili njegovim komponentama i realizaciju `dogodilo-se-prije` relacije. Kažemo da dretva **posjeduje lokot monitora** pridruženog objektu **između trenutka kada je zaključala lokot i otključala lokot**. Sve druge dretve koje istovremeno pokušaju pristupiti resursu će **čekati** (njihovo izvršavanje će biti pauzirano) **dok se lokot ne oslobodi**. Sinkronizirane metode **automatski dohvaćaju lokot monitora pridruženog objektu** a otpuštaju ga nakon poziva `return` (čak i ako je prijavljena iznimka).

Sinkronizacija monitorima pridruženim objektima

```
1  public class Monitori implements Runnable{
2     static int b1=0, b2=0;
3      int id;
4      Object l1 = null, l2 = null;
5      Monitori(int _id){  id = _id; }
6      Monitori(int _id, Object _l1, Object _l2){
7          id = _id; l1 = _l1; l2 = _l2;
8      }
9     public static void reset(){ b1 = b2  = 0; }
10    public void inc1(){ b1++; }
11    public void inc2(){ b2++; }
12
13    public void run(){
14       if(id == 0){
15        for(int i=0;i<100000;i++){ inc1(); }
16        for(int i=0;i<100000;i++){ inc2(); } }
17       else{
18           for(int i=0;i<100000;i++){ inc2(); }
19           for(int i=0;i<100000;i++){ inc1(); }  }
```

Neispravno sinkronizirani program.

Sinkronizacija monitorima pridruženim objektima

```
1  public void run(){
2  if(id == 0){
3      synchronized(l1){
4        for(int i=0;i<100000;i++){ inc1(); } }
5
6      synchronized(l2){
7        for(int i=0;i<100000;i++){ inc2(); } }
8       }
9  else{
10    synchronized(l2){
11      for(int i=0;i<100000;i++){ inc2(); } }
12
13    synchronized(l1){
14      for(int i=0;i<100000;i++){ inc1(); } }
15      }
```

Ispravno sinkronizirani program.

Korištenje mehanizama za osiguravanje konzistentnosti pogleda na memoriju

```
1  public class KonzistentnostMemorijeVolatile implements
       Runnable{
2       private static volatile double broj;
3       private static volatile double start;
4
5      @Override
6          public void run() {
7              while (start == 0.0) { Thread.yield(); }
8              System.out.println(broj);
9          }
10  public static void main(String args[])throws
       InterruptedException{
11    new Thread(new KonzistentnostMemorijeVolatile()).start();
12                  broj = 42.23;
13                  start = 2.0;  }
14     }
```

Volatile varijable.

Korištenje mehanizama za osiguravanje konzistentnosti pogleda na memoriju

U gornjem primjeru, ukoliko nebi koristili `volatile` varijable *broj* i *start* (koje osiguravaju atomarno čitanje i pisanje vrijednosti), novo kreirana dretva bi potencijalno dugo čekala u petlji dok se ne osvježi vrijednost varijable u *cache* memoriji (moglo bi se dogoditi i da završi u beskonačnoj petlji). Također je moguće da ispiše vrijednost 0.0.

Pisanje i čitanje (dakle sama operacija, ne operacije nad varijablama) je atomarna za sve varijable referenciranog tipa i varijable svih osnovnih tipova osim `long` i `double`. Čitanje i pisanje je atomarno za sve `volatile` varijable.

Zadatak - dz

Zadatak 1

Napišite višedretveni program koji će generirati vektor od n realnih brojeva (koristite klasu `java.util.Random`), zatim pokrenuti k dretava i korištenjem tih k dretava uz ravnomjerno raspoređene zadatke izbrojati broj pozitivnih, negativnih elemenata i broj nula u zadanom vektoru.