Višenitnost u Javi

AUDITORNE VJEŽBE

Sadržaj

Primjer paralelnog izvođenja tri niti

Primjer korištenja zajedničkog resursa bez sinkronizacije

Primjer korištenja zajedničkog resursa sa sinkronizacijom

Primjer programa za zauzimanje sjedala u kino dvorani

Primjer paralelnog izvođenja tri niti

```
public class TVProgramRunnableNit implements Runnable {

    private Thread nit;

    public TVProgramRunnableNit(String nazivPrograma) {
        nit = new Thread(this, nazivPrograma);
    }

    public void start() {
        nit.start();
    }

    …
```

Primjer paralelnog izvođenja tri niti

```
    @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Prebačeno na program '" + Thread.currentThread().getName()
                    + "' " + i + ". put.");

            try {
                Thread.sleep((int) Math.random() * 1000);
            }
            catch(InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        System.out.println("Završen program '" + Thread.currentThread().getName() + "'!");
    }
}
```

Primjer paralelnog izvođenja tri niti

```
public class GledanjeTVPrograma {

   private static final int BROJ_NITI = 3;

   public static void main(String[] args) {

      TVProgramRunnableNit prvaNit = new TVProgramRunnableNit("Utakmica Real - Barcelona");
      TVProgramRunnableNit drugaNit = new TVProgramRunnableNit("Turska sapunica");
      TVProgramRunnableNit trecaNit = new TVProgramRunnableNit("Dokumentarac");

      ExecutorService executorService = Executors.newFixedThreadPool(BROJ_NITI);
      executorService.execute(prvaNit);
      executorService.execute(drugaNit);
      executorService.execute(trecaNit);

      executorService.shutdown();
   }
}
```

Primjer paralelnog izvođenja tri niti

```
Prebačeno na program 'Turska sapunica' 1. put.
Prebačeno na program 'Dokumentarac' 1. put.
Prebačeno na program 'Utakmica Real - Barcelona' 1. put.
Prebačeno na program 'Dokumentarac' 2. put.
Prebačeno na program 'Utakmica Real - Barcelona' 2. put.
Prebačeno na program 'Turska sapunica' 2. put.
Prebačeno na program 'Utakmica Real - Barcelona' 3. put.
Prebačeno na program 'Dokumentarac' 3. put.
Završen program 'Utakmica Real - Barcelona'!
Prebačeno na program 'Turska sapunica' 3. put.
Završen program 'Dokumentarac'!
Završen program 'Turska sapunica'!
```

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
public class SimpleArray // CAUTION: NOT THREAD SAFE!
{
   private static final SecureRandom generator = new SecureRandom();
   private final int[] array; // the shared integer array
   private int writeIndex = 0; // shared index of next element to write

   // construct a SimpleArray of a given size
   public SimpleArray(int size)
   {
      array = new int[size];
   }

   // used for outputting the contents of the shared integer array
   public String toString()
   {
      return Arrays.toString(array);
   }
```

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
public void add(int value) // add a value to the shared array
{
   int position = writeIndex; // store the write index

   try
   {
      // put thread to sleep for 0-499 milliseconds
      Thread.sleep(generator.nextInt(500));
   }
   catch (InterruptedException ex)
   {
      Thread.currentThread().interrupt(); // re-interrupt the thread
   }
   // put value in the appropriate element
   array[position] = value;
   System.out.printf("%s wrote %2d to element %d.%n",
      Thread.currentThread().getName(), value, position);
   ++writeIndex; // increment index of element to be written next
   System.out.printf("Next write index: %d%n", writeIndex);
}
```

}

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
public class ArrayWriter implements Runnable
{
   private final SimpleArray sharedSimpleArray;
   private final int startValue;

   public ArrayWriter(int value, SimpleArray array)
   {
      startValue = value;
      sharedSimpleArray= array;
   }
   @Override
   public void run()
   {
      for (int i = startValue; i < startValue + 3; i++)
      {
         sharedSimpleArray.add(i); // add an element to the shared array
      }
   }
} // end class ArrayWriter
```

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
public class SharedArrayTest
{
   public static void main(String[] arg)
   {
      // construct the shared object
      SimpleArray sharedSimpleArray = new SimpleArray(6);

      // create two tasks to write to the shared SimpleArray
      ArrayWriter writer1 = new ArrayWriter(1, sharedSimpleArray);
      ArrayWriter writer2 = new ArrayWriter(11, sharedSimpleArray);

      // execute the tasks with an ExecutorService
      ExecutorService executorService = Executors.newCachedThreadPool();
      executorService.execute(writer1);
      executorService.execute(writer2);

      executorService.shutdown();
```

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
        try
        {
            // wait 1 minute for both writers to finish executing
            boolean tasksEnded =
                executorService.awaitTermination(1, TimeUnit.MINUTES);

            if (tasksEnded)
            {
                System.out.printf("%nContents of SimpleArray:%n");
                System.out.println(sharedSimpleArray); // print contents
            }
            else
                System.out.println(
                    "Timed out while waiting for tasks to finish.");
        }
        catch (InterruptedException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Primjer korištenja zajedničkog resursa bez sinkronizacije

```
pool-1-thread-2 wrote 11 to element 0.
Next write index: 1
pool-1-thread-1 wrote  1 to element 0.
Next write index: 2
pool-1-thread-1 wrote  2 to element 2.
Next write index: 3
pool-1-thread-2 wrote 12 to element 1.
Next write index: 4
pool-1-thread-1 wrote  3 to element 3.
Next write index: 5
pool-1-thread-2 wrote 13 to element 4.
Next write index: 6

Contents of SimpleArray:
[1, 12, 2, 3, 13, 0]
```

Primjer korištenja zajedničkog resursa sa sinkronizacijom

```
public class SimpleArray
{
   private static final SecureRandom generator = new SecureRandom();
   private final int[] array; // the shared integer array
   private int writeIndex = 0; // index of next element to be written

   // construct a SimpleArray of a given size
   public SimpleArray(int size)
   {
      array = new int[size];
   }

   // used for outputting the contents of the shared integer array
   public synchronized String toString()
   {
      return Arrays.toString(array);
   }
} // end class SimpleArray
```

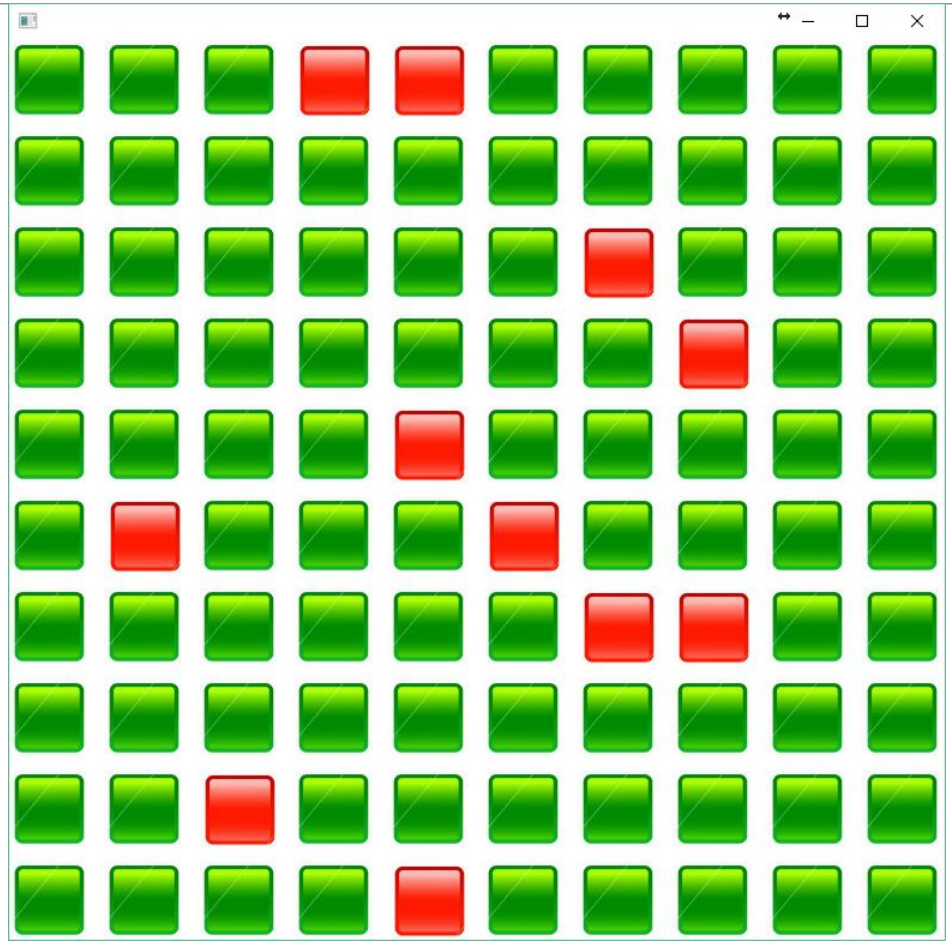
Primjer korištenja zajedničkog resursa sa sinkronizacijom

```
   // add a value to the shared array
   public synchronized void add(int value)
   {
      int position = writeIndex; // store the write index

      try
      {
         // in real applications, you shouldn't sleep while holding a lock
         Thread.sleep(generator.nextInt(500)); // for demo only
      }
      catch (InterruptedException ex)
      {
         Thread.currentThread().interrupt();
      }

      // put value in the appropriate element
      array[position] = value;
      System.out.printf("%s wrote %2d to element %d.%n",
         Thread.currentThread().getName(), value, position);

      ++writeIndex; // increment index of element to be written next
      System.out.printf("Next write index: %d%n", writeIndex);
   }
} // end class SimpleArray
```

Primjer korištenja zajedničkog resursa sa sinkronizacijom

```
pool-1-thread-1 wrote  1 to element 0.
Next write index: 1
pool-1-thread-1 wrote  2 to element 1.
Next write index: 2
pool-1-thread-1 wrote  3 to element 2.
Next write index: 3
pool-1-thread-2 wrote 11 to element 3.
Next write index: 4
pool-1-thread-2 wrote 12 to element 4.
Next write index: 5
pool-1-thread-2 wrote 13 to element 5.
Next write index: 6

Contents of SimpleArray:
[1, 2, 3, 11, 12, 13]
```

Primjer programa za zauzimanje sjedala u kinodvorani



Primjer programa za zauzimanje sjedala u kinodvorani

```
public class VisenitnostHelper {

        public static final int BROJ_REDOVA = 10;
        public static final int BROJ_STUPACA = 10;

        public static final int SJEDALO_SLOBODNO = 0;
        public static final int SJEDALO_ZAUZETO = 1;

        public static final int MAX_VRIJEME_CEKANJA = 10000;

        public static final String SLIKA_SLOBODNO_SJEDALO = "green.jpg";
        public static final String SLIKA_ZAUZETO_SJEDALO = "red.jpg";

        public static boolean isRezervacijaUTijeku = false;

        public static int[][] dvorana;
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
static {
        dvorana = new int[VisenitnostHelper.BROJ_REDOVA][VisenitnostHelper.BROJ_STUPACA];
}

public static void isprazniDvoranu() {
   for (int retci = 0; retci < BROJ_REDOVA; retci++) {
      for (int stupci = 0; stupci < BROJ_STUPACA; stupci++) {
         dvorana[retci][stupci] = SJEDALO_SLOBODNO;
      }
   }
}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public static synchronized float odrediPopunjenostDvorane() {

        int brojacPopunjenosti = 0;

        for (int retci = 0; retci < BROJ_REDOVA; retci++) {
                for (int stupci = 0; stupci < BROJ_STUPACA; stupci++) {
                        if (dvorana[retci][stupci] == SJEDALO_ZAUZETO) {
                                brojacPopunjenosti++;
                        }
                }
        }

        return (float) brojacPopunjenosti / (BROJ_REDOVA * BROJ_STUPACA);
}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public static synchronized void osvjeziPrikazDvorane() {
    Main.ocistiGrid();
    for (int retci = 0; retci < BROJ_REDOVA; retci++) {
        for (int stupci = 0; stupci < BROJ_STUPACA; stupci++) {
            if (dvorana[retci][stupci] == SJEDALO_SLOBODNO) {
                Main.sjedaloSlobodno(stupci, retci);
            }
            else {
                Main.sjedaloZauzeto(stupci, retci);
            }
        }
    }
}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public class OslobadjanjeMjestaNit implements Runnable {

        private int redniBroj;

        public OslobadjanjeMjestaNit(final int redniBrojNiti) {
                redniBroj = redniBrojNiti;
        }

        @Override
        public void run() {
                while(true) {
                        Random generator = new Random();
                        int redakSjedala = generator.nextInt(VisenitnostHelper.BROJ_REDOVA);
                        int stupacSjedala = generator.nextInt(VisenitnostHelper.BROJ_STUPACA);

                        try {
                                Thread.sleep(generator.nextInt(VisenitnostHelper.MAX_VRIJEME_CEKANJA));
                        } catch (InterruptedException e) {e.printStackTrace();}

                oslobodiSjedalo(redakSjedala, stupacSjedala);
        }
}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public synchronized void oslobodiSjedalo(final int redakSjedala, final int stupacSjedala) {
        while (VisenitnostHelper.isRezervacijaUTijeku == true) {
                try {
                        System.out.println("Redni broj #" + redniBroj + " - čekanje na oslobađanje sjedala „
                        + redakSjedala + ",  „ + stupacSjedala);
                        wait();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        VisenitnostHelper.isRezervacijaUTijeku = true;

        if (VisenitnostHelper.dvorana[redakSjedala][stupacSjedala] == VisenitnostHelper.SJEDALO_ZAUZETO) {
                VisenitnostHelper.dvorana[redakSjedala][stupacSjedala] = VisenitnostHelper.SJEDALO_SLOBODNO;
        }
        VisenitnostHelper.osvjeziPrikazDvorane();
        VisenitnostHelper.isRezervacijaUTijeku = false;

        notifyAll();
}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public class ZauzimanjeMjestaNit implements Runnable {

        private int brojNiti;

        public ZauzimanjeMjestaNit(final int redniBrojNiti) {
                brojNiti = redniBrojNiti;
        }
        @Override
        public void run() {
                while(true) {
                        Random generator = new Random();
                        int redakSjedala = generator.nextInt(VisenitnostHelper.BROJ_REDOVA);
                        int stupacSjedala = generator.nextInt(VisenitnostHelper.BROJ_STUPACA);
                        try {
                                Thread.sleep(generator.nextInt(VisenitnostHelper.MAX_VRIJEME_CEKANJA));
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        zauzmiSjedalo(redakSjedala, stupacSjedala);

}
```

Primjer programa za zauzimanje sjedala u kinodvorani

```
public class VisenitnostTest {

   private static final int BROJ_NITI_ZA_ZAUZIMANJE_DVORANE = 10;
   private static final int BROJ_NITI_ZA_OSLOBADJANJE_DVORANE = 10;

   public static void kreni() {
      VisenitnostHelper.isprazniDvoranu();

      ExecutorService executorServiceZauzimanje =
      Executors.newFixedThreadPool(BROJ_NITI_ZA_ZAUZIMANJE_DVORANE);
      for (int brojacNiti = 0; brojacNiti < BROJ_NITI_ZA_ZAUZIMANJE_DVORANE; brojacNiti++) {
         executorServiceZauzimanje.execute(new ZauzimanjeMjestaNit(brojacNiti + 1));
      }

      ExecutorService executorServiceOslobadjanje =
      Executors.newFixedThreadPool(BROJ_NITI_ZA_OSLOBADJANJE_DVORANE);
         for (int brojacNiti = 0; brojacNiti < BROJ_NITI_ZA_OSLOBADJANJE_DVORANE; brojacNiti++) {
         executorServiceOslobadjanje.execute(new OslobadjanjeMjestaNit(brojacNiti + 1));
      }
   }
}
```

Pitanja?