Systemy Rozproszone - Ćwiczenie 3

1 Współbieżny dostęp do obiektu

Poniżej znajduje się bardzo prosta klasa - licznik, mający 2 metody: `inc()` zwiększający licznik o 1 i `getValue()` zwracający aktualną wartość licznika. W ramach rozgrzewki napisz program, który będzie zwiększać licznik przez 2 sekundy, a po 2 sekundach wypisze wartość licznika. Do tego celu można np. wykorzytać `System.currentTimeMillis()`.

```
/* plik SimpleCounter.java */
public class SimpleCounter {
   private int value;

   public SimpleCounter() {
     value = 0;
   }

   public void inc() {
     value++;
   }

   public int getValue() {
     return value;
   }
}
```

Następnie stworzymy program, który utworzy 3 wątki i 1 obiekt SimpleCounter. W konstruktorze każdemu wątkowi przekazana zostanje referencja do obiektu SimpleCounter. Każdy proces zwiększa licznik 10 razy. Zastanów się co ostatecznie wypisze program na wyjściu, a następnie sprawdź to uruchomiając program.

```
public class SyncDemo extends Thread {
  SimpleCounter so;

  public SyncDemo(SimpleCounter obj) {
    so = obj;
  }
```

```
  public void run() {
    for (int i=0;i<10;i++) {
      so.inc();
    }
  }

  public static void main(String args[]) {
    SimpleCounter counter = new SimpleCounter();
    SyncDemo thread1 = new SyncDemo(counter);
    SyncDemo thread2 = new SyncDemo(counter);
    SyncDemo thread3 = new SyncDemo(counter);
    thread1.start();
    thread2.start();
    thread3.start();
    try {
      thread1.join();
      thread2.join();
      thread3.join();
    }
    catch (InterruptedException e) {
      System.out.println("Interrupted");
    }

    System.out.println(counter.getValue());
  }
}
```

Zwiększ ilość iteracji pętli w metodzie `run()` do 100, 1000, 100000. Przetestuj działanie programu dla różnej liczby wątków (od 1 do 10). Czy wyniki są zgodne z oczekiwaniami? Co mogło pójść nie tak? Zastanów się nad tym wspólnie z kolegą siedzącym obok Ciebie. Podpowiedź znajduje się na następnej stronie.

Co poszło nie tak?

Wszystkie 3 procesy wykonywały równolegle metodę `inc()`. Linia `value++` to tak naprawdę 3 instrukcje atomowe:

- skopiowanie zmiennej do rejestru: `LD R,value`
- zwiększ rejestr: `INC R`
- skopiowanie rejestru do zmiennej: `LD value,R`

Zastanów się jaki może wystąpić przeplot tych operacji atomowych podczas wykonywania metody `inc()` przez 2 procesy. Odpowiedź znajduje się na następnej stronie.

Niekorzystny przeplot operacji

```
/*
* przyklad na zwiekszanie o 1 przez 2 procesy (P0, P1) rownoczesnie
*
 P0: LD R, x                                    zaladuj x do rejestru
 P0: INC R                                      zwieksz rejestr
                P1 : LD R, x
                P1 : INC R
 P0: ST R,x                                     zapisz x do rejestru
                P1: ST R,x
*
*/
```

Problem równoczesnego dostępu do metody można rozwiązać poprzez użycie słowa `synchronized` w deklaracji metody. Poczytaj nt. synchronized i spróbuj naprawić licznik.

2 Sekcje krytyczne wewnątrz metody

Przykład:

```
public class AdvancedCounter {
  private int value1 = 0;
  private int value2 = 0;
  private Object lock1 = new Object();
  private Object lock2 = new Object();


  public void inc1() {
    synchronized(lock1) {
      value1++;
    }
  }

  public void inc2() {
    synchronized(lock2) {
      value2++;
    }
  }

  public int get1() {
    return value1;
  }

  public int get2() {
    return value2;
```

```
  }
}
```

3 Sekcje krytyczne - piaskownica

Poeksperymentuj z poniższym programem używając słowa kluczowego `synchronized` zarówno dla metod jak i bloków kodu.

```
class Shared {
    public void foo(String threadName, long time) {
        System.out.println(threadName +
          " is running foo() for "+time+" ms, ct:" +
          System.currentTimeMillis());
        long t = System.currentTimeMillis();
        while (System.currentTimeMillis()-t < time) {
            Math.atan(System.currentTimeMillis());
        }
    }

    public void bar(String threadName, long time) {
        System.out.println(threadName +
          " is running bar() for "+time+" ms, ct:" +
          System.currentTimeMillis());
        long t = System.currentTimeMillis();
        while (System.currentTimeMillis()-t < time) {
            Math.atan(System.currentTimeMillis());
        }
    }
}

class ThreadA extends Thread {
    Shared shared;
    ThreadA(Shared instance) {
        this.shared = instance;
    }

    public void run() {
        shared.foo("A", 5000);
        shared.bar("A", 1000);
        System.out.println("A is done");
    }
}

class ThreadB extends Thread {
    Shared shared;
```

```
    ThreadB(Shared instance) {
        this.shared = instance;
    }

    public void run() {
        shared.foo("B", 1000);
        shared.bar("B", 2000);
        System.out.println("B is done");
    }
}

public class JavaApplication1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Shared shared = new Shared();
        ThreadA th1 = new ThreadA(shared);
        ThreadB th2 = new ThreadB(shared);
        long t = System.currentTimeMillis();
        th1.start();
        th2.start();

        try {
            th1.join();
            th2.join();
        }
        catch (InterruptedException e) {
            System.out.println(e);
        }
        System.out.println("done in " + (System.currentTimeMillis() - t) + " ms"
    }
}
```

4 Zakleszczenie

Dlaczego dochodzi do zakleszczenia?

```
public class DeadlockDemo1 {

  static class Friend {

    private final String name;
```

```
    public Friend(String name) {
      this.name = name;
    }

    public String getName() {
      return this.name;
    }

    public synchronized void bow(Friend bower) {
      System.out.format("%s: %s has bowed to me!%n",
              this.name, bower.getName());
      bower.bowBack(this);
    }

    public synchronized void bowBack(Friend bower) {
      System.out.format("%s: %s has bowed back to me!%n",
              this.name, bower.getName());
    }
  }

  public static void main(String[] args) {
    final Friend alphonse = new Friend("Alphonse");
    final Friend gaston = new Friend("Gaston");

    new Thread(new Runnable() {
      public void run() {
        alphonse.bow(gaston);
      }
    }).start();

    new Thread(new Runnable() {
      public void run() {
        gaston.bow(alphonse);
      }
    }).start();
  }
}
```

5 Zadania

Na razie nic....