Semaphore Beispiel

```
import java.util.concurrent.Semaphore;

public class SemaphoreSum extends Thread {
	//Attribute
	public static int[] array = new int[1000000]; //in 4 geteilt
	public static long sum=0;  //Gesamtsumme
	private int lower;         //Untergrenze
	private int upper;         //Obergrenze
	private int id;            //ID des Threads (für Output)


	public SemaphoreSum(int[] a, int l, int u, int id) {
		array=a;
		lower=l;
		upper=u;
		this.id = id;
	}

	public static void main(String[] args) throws InterruptedException {
		for (int i = 0; i < array.length; i++) {
			array[i]=i;
		}
		//kreiere die Worker
		int k=array.length;
		SemaphoreSum ss1 = new SemaphoreSum(array, 0, (k/4)-1, 1);
		SemaphoreSum ss2 = new SemaphoreSum(array, (k/4), (k/2)-1, 2);
		SemaphoreSum ss3 = new SemaphoreSum(array, k/2, (3*k/4)-1, 3);
		SemaphoreSum ss4 = new SemaphoreSum(array, 3*k/4, k-1, 4);
		//Starte die Worker

		long start = System.currentTimeMillis();


		ss1.start();


		ss2.start();


		ss3.start();


		ss4.start();
		while //your condition here:(                          ) {
			//do nothing
		}
		long end = System.currentTimeMillis();
		System.out.println("Gesammtsumme parallel: " + sum);
		System.out.println("Dauer parallele Version in ms: " + (end
		- start));
	}
```

```
	public void run() {
		long psum = 0;		//Teilsumme
		for (int i = lower; i <= upper; i++) {
			psum+=array[i];
		}
		System.out.println("Thread " + id + " ist fertig und hat
		berechnet: " + psum);

		//Critical Section









		sum+=psum;
		//End Critical Section



	}

}
```

Verwende 2 Semaphores:

- Eins zum schützen des Updates von sum.
- Eins zum herausfinden, dass alle Threads fertig sind und somit sum steht.
- Tipp: `availablePermits()` gibt an, wieviele Permits vorhanden sind… am Ende sollten dies 4 sein.

Meine Lösung

```
package semaphore;

import java.util.concurrent.Semaphore;

public class SemaphoreSum extends Thread {
	//Attribute
	public static int[] array = new int[1000000]; //wird in 4 Teile
aufgeteilt
	public static long sum=0;  //Gesamtsumme
	private int lower;         //Untergrenze des zu bearbeitenden
Stücks
	private int upper;         //Obergrenze
	private int id;            //ID des Threads (für Output)
	public static Semaphore sem2 = new Semaphore(4,true);
	public static Semaphore sem1 = new Semaphore(1,true); //Mutex
Semaphore

	public static void main(String[] args) throws
InterruptedException {
		for (int i = 0; i < array.length; i++) {
			array[i]=i;
		}
		//kreiere die Worker
		int k=array.length;
		SemaphoreSum ss1 = new SemaphoreSum(array, 0, (k/4)-1, 1);
		SemaphoreSum ss2 = new SemaphoreSum(array, (k/4), (k/2)-1, 2);
		SemaphoreSum ss3 = new SemaphoreSum(array, k/2, (3*k/4)-1, 3);
		SemaphoreSum ss4 = new SemaphoreSum(array, 3*k/4, k-1, 4);
		//Starte die Worker

		long start = System.currentTimeMillis();
		sem2.acquire();
		ss1.start();
		sem2.acquire();
		ss2.start();
		sem2.acquire();
		ss3.start();
		sem2.acquire();
		ss4.start();
		while (sem2.availablePermits()!=4) {
			//do nothing
		}
		long end = System.currentTimeMillis();
		System.out.println("Gesammtsumme parallel: " + sum);
		System.out.println("Dauer parallele Version in ms: " + (end
- start));
		long seqstart = System.currentTimeMillis();
		long sumseq = seq();
		long seqend = System.currentTimeMillis();
		System.out.println("Gesammtsumme sequentiell: " + sumseq);
```

```
            System.out.println("Dauer sequentielle Version in ms: " +
(seqend-seqstart));
      }

      private static long seq() {
            long sumseq= 0;
            for (int i = 0; i < array.length; i++) {
                  sumseq+=array[i];
            }
            return sumseq;
      }

      public SemaphoreSum(int[] a, int l, int u, int id) {
            array=a;
            lower=l;
            upper=u;
            this.id = id;
      }

      //überschreiben der run Methode von Thread
      public void run() {
            long psum = 0;        //Teilsumme
            for (int i = lower; i <= upper; i++) {
                  psum+=array[i];
            }
            System.out.println("Thread " + id + " ist fertig und hat
berechnet: "
                        + psum);
            /*Dieser Teil ist heikel, es könnte sein, dass mehrere
Threads
            zur selben Zeit auf sum zugreifen möchten. Darum müssen wir
            diesen Block synchronisieren.*/
            try {
                  sem1.acquire();
            } catch (InterruptedException e) {
            }
            sum+=psum;
            sem1.release();
            sem2.release();
      }

}
```