Análisis y diseño de software

dit
UPM

Tema 3: Concurrencia
java library synchronizers

José A. Mañas

26.4.2018

ETSIT UPM

synchronizers

- Java provides some classes
  for common special-purpose synchronization
  - Semaphore
  - Lock
  - BlockingQueue<E> (aka producers-consumers)
  - SynchronousQueue<E>

  - CountDownLatch
  - CyclicBarrier
  - Exchanger<V>
  - ~~Phaser~~

package java.util.concurrent

Semaphore

- Semaphore(int permits)
- void acquire(int permits)
  - espera hasta que hay # permisos y los retira
- void release(int permits)
  - devuelve # permisos y avisa a los que esperan

- void acquire() { acquire(1); }
- void release() { release(1); }

semaphores

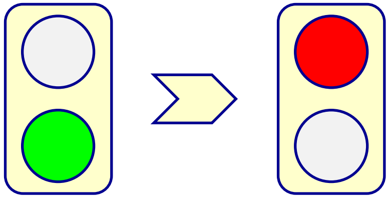
- Dijkstra, one of the inventors of semaphores, used P and V.
- The letters come from the Dutch words Probeer (try) and Verhoog (increment).
- Over seinpalen
  - https://cs.nyu.edu/~yap/classes/os/resources/EWD74.pdf

uso semáforo binario

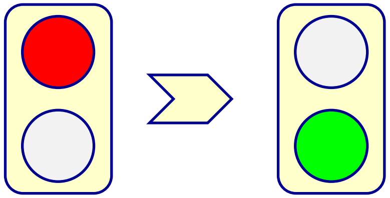
```
semaforo.acquire();

    ... operaciones ...

semaforo.release();
```

```
semaforo.acquire();
try {
    ... operaciones ...
} finally {
  semaforo.release();
}
```

acquire



release



uso de los semáforos

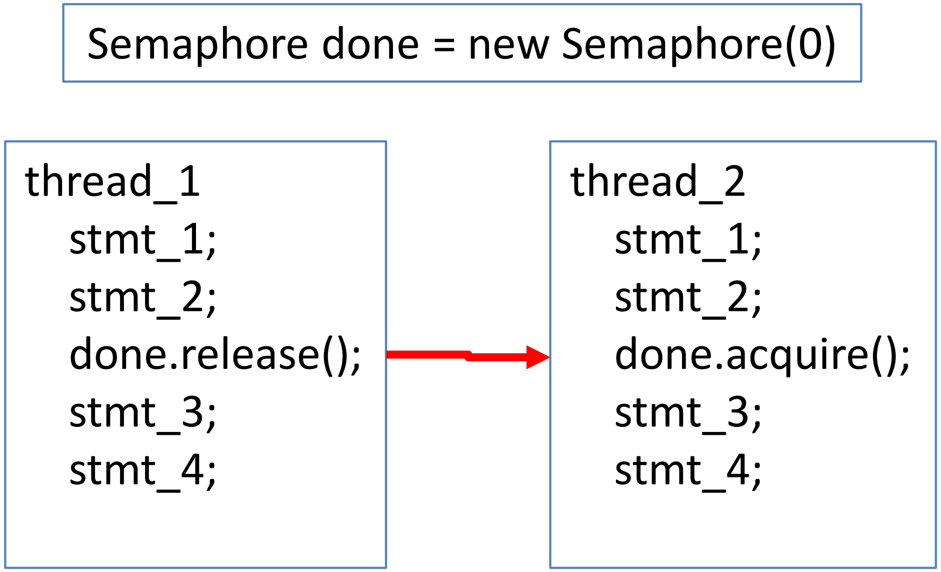
1. limitar el número de threads en la zona crítica

```
Semaphore sm = new Semaphore(5) # Max: 5-threads

InputStream fetch_page(String ref):
    sm.acquire();
    try {
        URL url = new URL(ref);
        return url.openStream();
    } finally {
        sm.release();
    }
```

```
Semaphore semaphore = new Semaphore(N)

semaphore.acquire();
try {
    … zona crítica …
} finally {
    semaphore.release();
}
```

uso de los semáforos

2. coordinar threads



```
Semaphore done = new Semaphore(0)
```

```
thread_1
   stmt_1;
   stmt_2;
   done.release();
   stmt_3;
   stmt_4;
```

```
thread_2
   stmt_1;
   stmt_2;
   done.acquire();
   stmt_3;
   stmt_4;
```

ejemplo

semáforo con N permisos

```
Semaphore contador = new Semaphore(0);
List<Threat> tareas = new ArrayList<Threat>();

Semaphore contador = new Semaphore(0);

for (Tarea tarea : tareas)
    tarea.start();

// espera a que todas acaben
contador.acquire(tareas.size());
```

```
public class Tarea extends Thread {
    private Semaphore contador;

    Tarea(Semaphore contador){ this.contador = contador;}

    public void run() {
        // hace su tarea
        contador.release();
    }
}
```

ejemplo

semáforo con N permisos

```
Semaphore contador = new Semaphore(0);
List<Threat> tareas = new ArrayList<Threat>();

Semaphore bandera = new Semaphore(0);

for (Tarea tarea : tareas)
    tarea.start();

// preparados, listos, ¡ya!
contador.release(tareas.size());
```

```
public class Tarea extends Thread {
    private Semaphore bandera;

    Tarea(Semaphore bandera){ this.bandera = bandera;}

    public void run() {
        contador.acquire();    // a la señal
        // hace su tarea
    }
}
```

bounded buffer

ejemplo

```
public class Buffer<E> {
   private final List<E> queue = new ArrayList<>(size);

   private final Semaphore haySitio = new Semaphore(size);
   private final Semaphore hayDatos = new Semaphore(0);
   private final Semaphore mutex = new Semaphore(1);
```

```
public void put(E s)
    throws InterruptedException {
     haySitio.acquire();
     mutex.acquire();
     try {
        queue.add(s);
     } finally {
        mutex.release();
        hayDatos.release();
     }
  }
```

```
public E take()
  throws InterruptedException {
   hayDatos.acquire();
   mutex.acquire();
   try {
      return queue.remove(0);
   } finally {
      mutex.release();
      haySitio.release();
   }
}
```

errores (fragilidad)

ejemplo

```
public void put(E s) throws InterruptedException {
    mutex.acquire();
    haySitio.acquire();
    try {
        queue.add(s);
    } finally {
        mutex.release();
        hayDatos.release();
    }
  }
```

```
public E take() throws InterruptedException {
    mutex.acquire();
    hayDatos.acquire();
    try {
        return queue.remove(0);
    } finally {
        mutex.release();
        haySitio.release();
    }
}
```

Lock

- class ReentrantLock implements Lock
- class ReadWriteLock implements Lock

- void lock()
- void unlock()

- Condition newCondition()

estado compartido protegido

```
public class Contador {
    private int cuenta = 0;
    private final Lock LOCK = new ReentrantLock();
```

```
    public int incrementa(int v) {
        try {
            LOCK.lock();
            cuenta += v;
            return cuenta;
        } finally {
            LOCK.unlock();
        }
    }
```

```
    public int decrementa(int v) {
        try {
            LOCK.lock();
            cuenta -= v;
            return cuenta;
        } finally {
            LOCK.unlock();
        }
    }
```

ReadWriteLock

- class ReentrantReadWriteLock
  implements ReadWriteLock
- Lock readLock()
- Lock writeLock()

ejemplo

Condition

- colas dentro de un cerrojo

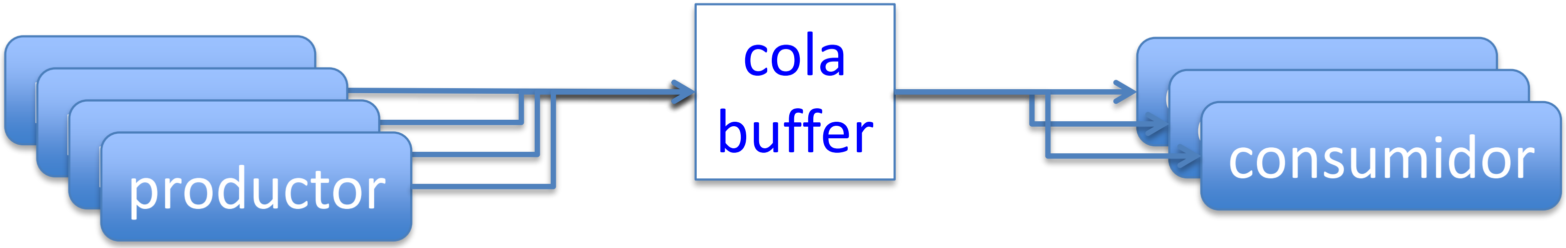
```
private Lock lock = new ReentrantLock();
private Condition isEmpty = lock.newCondition();
private Condition isFull = lock.newCondition();
```

```
public void put(E x)
        throws InterruptedException {
    lock.lock();
    while (data.size() >= SIZE)
        isFull.await();
    data.add(x);
    isEmpty.signalAll();
    lock.unlock();
}
```

```
public E get()
        throws InterruptedException {
    lock.lock();
    while (data.isEmpty())
        isEmpty.await();
    E value = data.remove(0);
    isFull.signalAll();
    lock.unlock();
    return value;
}
```

BlockingQueue<E>

- BlockingQueue<E>(int max)
- void put(E e)
  - lo añade al buffer si cabe; si no, espera
- E take()
  - saca si hay algo en el buffer; si no, espera



productores – consumidores

bounded buffer pattern

SynchronousQueue<E>

- SynchronousQueue<E>()
  - A blocking queue in which each insert operation must wait for a corresponding remove operation by another thread, and vice versa.
  - A synchronous queue does not have any internal capacity, not even a capacity of one.
- void put(E e)
- E take()
  - rendezvous asimétrico
    - the sender blocks until the message is received

CountDownLatch

- CountDownLatch(int count)
  - A CountDownLatch is initialized with a given count. The await methods block until the current count reaches zero due to invocations of the countDown() method, after which all waiting threads are released and any subsequent invocations of await return immediately.
- void await()
- void countDown()

CountDownLatch

ejemplo

```
    CountDownLatch startSignal = new CountDownLatch(1);
    CountDownLatch doneSignal = new CountDownLatch(N);

    for (Worker worker: …)
      worker.start();

    doSomethingElse();
    startSignal.countDown();
    doSomethingElse();
    doneSignal.await();
   }
  }
```

```
class Worker extends Thread {
  public void run() {
   try {
    startSignal.await();
    doWork();
    doneSignal.countDown();
   } catch (InterruptedException ex) {}
   // return;
  }
```

CyclicBarrier

- CyclicBarrier(int required)
  - A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point.
- void await()

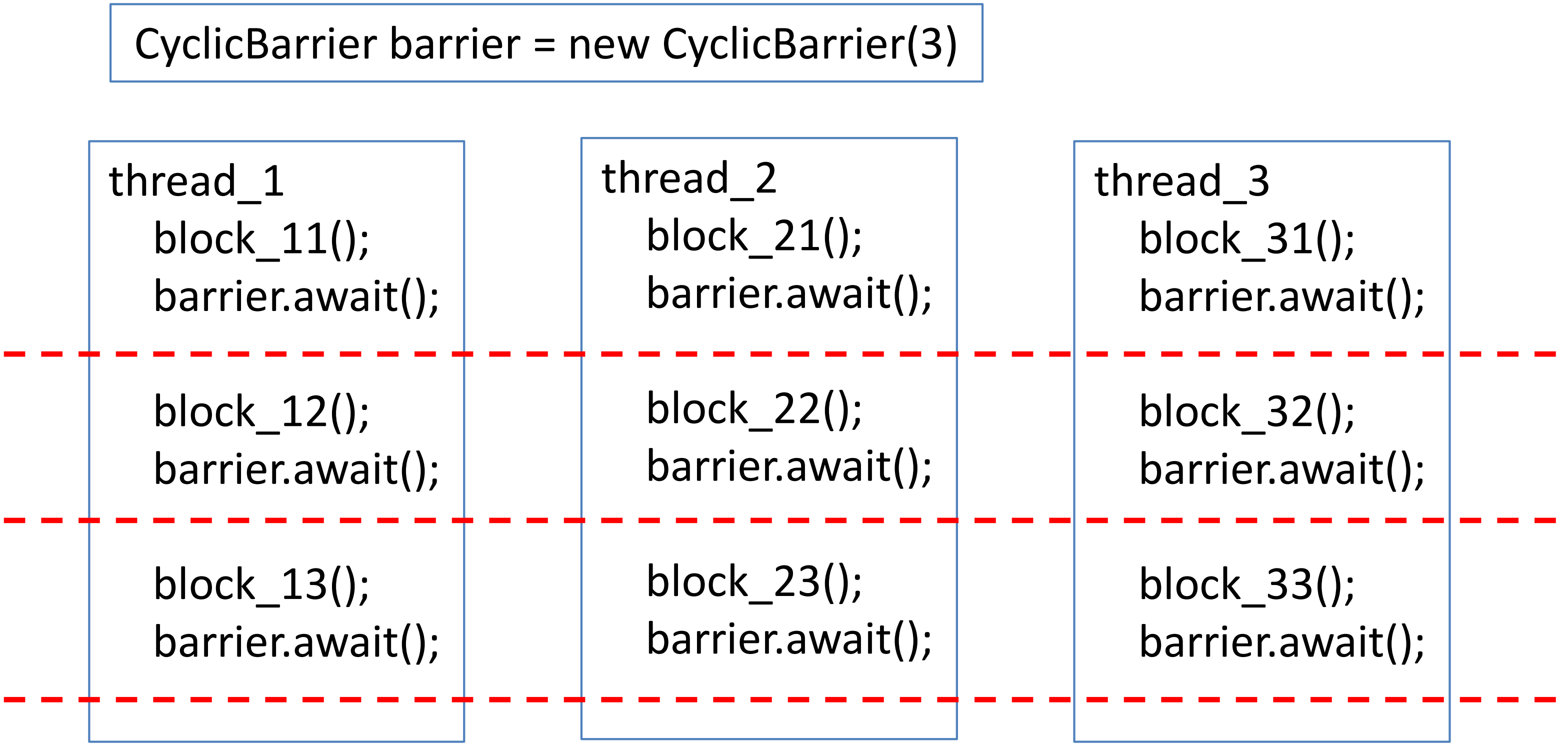
versión con código a ejecutar cuando se abre

- CyclicBarrier(int parties, Runnable action)

ejemplo

CyclicBarrier

coordinar threads



```
CyclicBarrier barrier = new CyclicBarrier(3)
```

```
thread_1
    block_11();
    barrier.await();

    block_12();
    barrier.await();

    block_13();
    barrier.await();
```

```
thread_2
    block_21();
    barrier.await();

    block_22();
    barrier.await();

    block_23();
    barrier.await();
```

```
thread_3
    block_31();
    barrier.await();

    block_32();
    barrier.await();

    block_33();
    barrier.await();
```

Exchanger<V>- rendezvous

- Exchanger<V>()
  - A synchronization point at which threads can pair and swap elements within pairs. Each thread presents some object on entry to the exchange method, matches with a partner thread, and receives its partner's object on return. An Exchanger may be viewed as a bidirectional form of a SynchronousQueue.
- V exchange(V x)
  - rendezvous simétrico
    - the sender blocks until the message is received

synchronous bounded buffer

ejemplo

```
 private static MyBuffer buffer1 = new MyBuffer();
 private static MyBuffer buffer2 = new MyBuffer();
 private static Exchanger<MyBuffer> exchanger = new Exchanger<>();

 MyProducer producer = new MyProducer(buffer1);
MyConsumer consumer = new MyConsumer(buffer2);
```

```
private static class MyBuffer {
    private static final int SIZE = 10;
    private Integer[] data = new Integer[SIZE];
    private int at= 0;

    public boolean isEmpty() { return at == 0;}
    public boolean isFull() { return at == SIZE;}
    public void write(Integer n) { data[at++]= n;}
    public Integer read() { return data[--at];}
  }
```

synchronous bounded buffer

ejemplo

```
class MyProducer {
    private MyBuffer buffer;

    public void run() {
        … … …
        buffer.write(n);
        if (buffer.isFull())
            buffer = exchanger.exchange(buffer);
```

```
private static class MyConsumer extends Thread {
    private MyBuffer buffer;

    public void run() {
        …. … …
        if (buffer.isEmpty())
            buffer=exchanger.exchange(buffer);
        Integer x = buffer.read();
```

atomic variables

- java.util.concurrent.atomic
- AtomicInteger

```
class AtomicCounter {
    private AtomicInteger c = new AtomicInteger(0);

    public void increment() {
        c.incrementAndGet();
    }

    public void decrement() {
        c.decrementAndGet();
    }

    public int value() {
        return c.get();
    }
```