Hilos en Java

Luis Daniel Benavides Navarro,19-9-2017

Varios de los ejemplos de código son inspirados en los encontrados en el tutorial de Java encontrado en:
https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

Modelo General

- Virtualiza ejecución

- Comparte memoria

Dos maneras de crear hilos

```java
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

```java
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Esperando por la respuesta de un hilo (join)

```
public class HelloRunnableWithJoin implements Runnable{

    public void run() {
        try {
            System.out.println("Hello from a thread! time: " + LocalDateTime.now());
            Thread.sleep(4000);
        } catch (InterruptedException ex) {
            Logger.getLogger(HelloRunnableWithJoin.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String args[]) {
        Thread t= new Thread(new HelloRunnableWithJoin());
        t.start();
        try {
            t.join();
        } catch (InterruptedException ex) {
            Logger.getLogger(HelloRunnableWithJoin.class.getName()).log(Level.SEVERE, null, ex);
        }
        System.out.println("Hello from main thread! time: " + LocalDateTime.now());
    }
}
```

Métodos Sincronizados

- Usted puede tener un método sincronizado o un bloque sincronizado

- Un método sincronizado implica que un solo hilo puede estar accediendo a los métodos sincronizados de ese objeto.

- Ejemplo, dos métodos sincronizados en el mismo objeto

```
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());
}
```

Seguros y sincronización

- El mecanismo de sincronización está construido alrededor de los seguros intrínsecos.
- Básicamente, un hilo debe adquirir el seguro de un objeto para ejecutar un método sincronizado, si no lo puede adquirir espera por la liberación del seguro.
- Ejemplo, se mejora la concurrencia si los contadores del objeto no son usados nunca juntos:

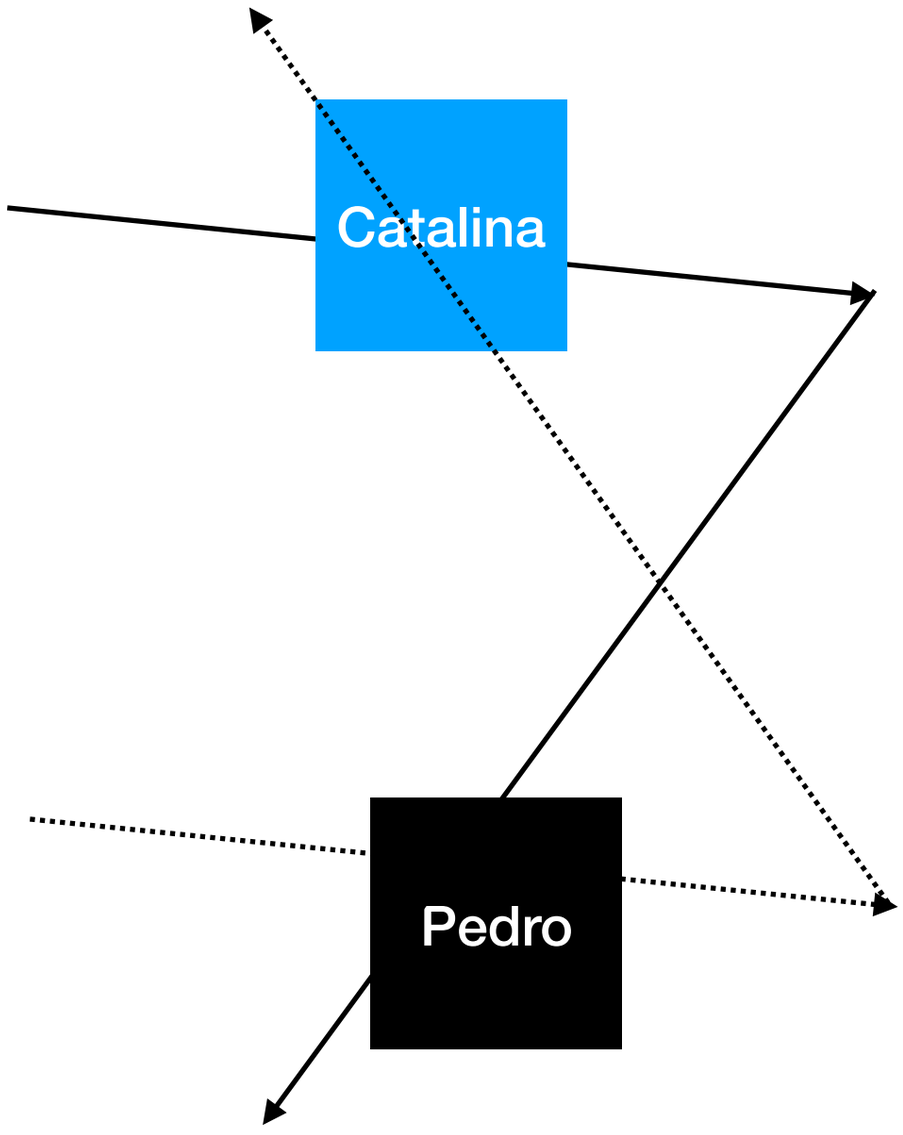
```
public class SynchronizedStatement {
    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        synchronized(lock1) {
            c1++;
        }
    }

    public void inc2() {
        synchronized(lock2) {
            c2++;
        }
    }
}
```

Liveness y Deadlock

- La habilidad de una aplicación de ejecutarse de una manera temporalmente correcta se denomina Liveness.
- Uno de los errores que más afectan está propiedad es el Deadlock

```
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();
}
```


Catalina
Pedro

Bloques con guarda

- Para sincronizar acciones dos objetos pueden usar bloque con guarda.
- La guarda es una condición, y uno de los thread solo procede cuando la condición se cumple.
- Ejemplo, una guarda ineficiente hace un loop hasta que la condición se cumple

```
public void guardedJoy() {
    // Simple loop guard. Wastes
    // processor time. Don't do this!
    while(!joy) {}
    System.out.println("Joy has been achieved!");
}
```

wait y notifyAll

- El método Object.wait suspende la ejecución de un hilo, y la reinicia cuando recibe una notificación de que un evento importante paso.
- El método Object.notifyAll le notifica a todos los hilos que algo importante pasó
- El método Object.notify le notifica a un hilo cualquiera que algo pasó

```
public synchronized void guardedJoy() {
    // This guard only loops once for each special event, which may not
    // be the event we're waiting for.
    while(!joy) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    System.out.println("Joy and efficiency have been achieved!");
}

. . .
public synchronized notifyJoy() {
    joy = true;
    notifyAll();
}
```

- Cuando se invoca wait, el objeto libera el seguro y espera
- Siempre invoque wait en un ciclo que revise si la condición de guarda se cumple

Objetos inmutables

- Crear objetos inmutables, es decir que su estado no puede cambiar, se considera una buena estrategia de código seguro

- Los programadores subestiman esta técnica. Porque creen que crear objetos es muy costoso, pero los beneficios de ser inmutables superan este costo.

- Cómo crearlos
  - No cree métodos para modificar el estado (“setter”)
  - Todos los campos deben ser finales y privados
  - No permita que las subclases sobre-escriban métodos. Puede ser declarando la clase final o creando constructores privados y método fábrica.
  - Si los campos de instancia tiene referencias a objetos, no permitan que se cambien
    - No provea métodos para modificar los objetos mutables
    - No comparta referencias a objetos mutables. Cree copias si es necesario.

Concurrencia de alto nivel

- Java provee un API de alto nivel para manejar concurrencia
    - Executors, proveen clases para manejar por ejemplo pools de hilos
    - Objetos Lock, soportan idiomas de lock más sofisticados
    - Colecciones concurrentes, estructuras de datos que son seguras en aplicaciones concurrentes
    - Variables atómicas, tipos de variables que solo permiten ser modificadas por un hilo a la vez (minimizan sincronización).

Interfaces Executor

- **Executor:**
  - un solo método execute. soporta lanzamiento de nuevas tareas.
  - e.execute(r); // r es runnable
- **ExecutorService implements Executor**:
  - provee método *submit* que retorna un futuro. Recibe objetos *Runnable* y *Callable*.
  - Puede recibir grandes colecciones de objetos
  - *Callable* y provee método de gestión de tareas.
- **ScheduledExecutorService implements ExecutorService:**
  - Entrega el método schedule que inicia tareas en periodos especificados.

Pool de Hilos

- Patrón de programación:
  - administra un conjunto limitado de hilos para no sobrecargar el sistema.
  - Asigna los hilos de acuerdo a una estrategia dada
- Al API provee Executors que implementan el patrón
  - Para crear gestores de hilos se pueden invocar los métodos fábrica in *java.util.concurrent.Executors*:
    - newFixedThreadPool, pool de hilos de tamaño fijo
    - newCachedThreadPool, pool de hilos variable para muchas tareas paralelas de corta duración
    - newSingleThreadExecutor, una sola tarea a la vez.

¿Preguntas?