# Programación Concurrente
# Prácticas

## Monitores

# 1. Implementación de Monitores en Java

| | Hoare | Java |
|---|---|---|
| **Definir un monitor** | `Monitor m;` | **Las funciones del monitor se ejecutan en E.M. haciendo uso de *Locks***<br><br>`import java.util.concurrent.locks;`<br>`Lock m = new ReentrantLock();`<br>`public void func () {`<br>`  m.lock();`<br>`  ...`<br>`  m.unlock();`<br>`}` |
| **Variables de condición** | `Condition c;`<br>`c.delay();`<br>`c.resume();` | **La vble. de condición se asocia a un Lock (condición de sincronización)**<br><br>`Condition c = m.newCondition();`<br>`c.await();`<br>`c.signal();` |

## java.util.concurrent.locks

### Interfaces
- `Lock`
- `Condition`

### Implementaciones
- `ReentrantLock`
- `<ninguna>`

### Creación de objetos (instancias)
- `Lock x = new ReentrantLock();`
- `Condition c = x.newCondition();`

**`Condition`** no permite la creación de instancias (objetos). Esta ligada a la clase Lock y se usa una función de esta interfaz para crear un objeto

# 1. Utilización de monitores (Productor/Consumidor)

```
import java.util.concurrent.locks.*;
public class Buffer {
        Lock monitor = new ReentrantLock();
        Condition lleno = monitor.newCondition();
        Condition vacio = monitor.newCondition();

        public void put (int elemento) {
                monitor.lock();
                try {   while (cuantos == N) lleno.await();
                        /* s.c. INSERTAR */
                        vacio.signal();
                } finally
                        monitor.unlock();
        }
        public int get () {
                monitor.lock();
                try {   while (cuantos == 0) vacio.await();
                        /* s.c. EXTRAER */
                        lleno.signal();
                } finally
                        monitor.unlock();
        }
}
```

- La semántica del *signal (resume)* es **Desbloquear y Continuar** (DC). El proceso desbloqueador continúa ejecutando y cuando el desbloqueado empieza su ejecución no podemos asegurar que la condición que le hizo bloquearse se haya dejado de cumplir. Por tanto debe reevaluar su condición de nuevo (`while`).

- Se debe asegurar que la salida del monitor siempre se hace de forma correcta ( ejecución del `unlock`) . Por eso el bloque `try…finally`.

- `await()` lanza una excepción de la clase `InterruptedException` que hay que capturar o relanzar.