



# Programación Concurrente y Distribuida

---

## Práctica 2

## Sincronización en Java



# Sincronización en Java

## ■ EXCLUSIÓN MUTUA

- Para conseguir la exclusión mútua sobre una parte del código se usa **synchronized**
- **synchronized** puede ser aplicado sobre un método completo de una clase, lo que hace exclusión de dicho método con cualquier otro método de la clase que también esté marcado como **synchronized**.

```
class practica4 {
```

```
    public synchronized void metodo1(){
```

```
        // Esto se ejecuta en exclusión mútua
```

```
    }
```

```
    public synchronized void metodo2(){
```

```
        // Esto se ejecuta en exclusión mútua
```

```
    }
```

```
    public synchronized void metodo3(){
```

```
        // Esto puede ser concurrente
```

```
    }
```

Sólo es posible 1 hilo en cualquiera de ellas

Cualquier cantidad de hilos, incluso a la vez que el de los metodos 1 o 2



# Sincronización en Java

## EXCLUSIÓN MUTUA

- **synchronized** siempre está asociado a un objeto, de forma que la exclusión mutua sólo se realiza entre el código sincronizado sobre el mismo objeto, y no entre código sincronizado en objetos distintos.
- Cuando se sincroniza un método de una clase, el objeto sobre el que se sincroniza es **this**.

```
class practica4 {  
  
    public synchronized void metodo1(){  
        // Esto se ejecuta en exclusión mútua  
    }  
    public synchronized void metodo2(){  
        // Esto se ejecuta en exclusión mútua  
    }  
}
```

```
class practica4b {  
  
    public synchronized void metodo1b(){  
        // Esto se ejecuta en exclusión mútua  
    }  
    public synchronized void metodo2b(){  
        // Esto se ejecuta en exclusión mútua  
    }  
}
```

Sólo es posible 1  
hilo en cualquiera  
de ellas

Pueden ser  
concurrentes

Sólo es posible 1  
hilo en cualquiera  
de ellas



# Sincronización en Java

## EXCLUSIÓN MUTUA

- **synchronized** siempre está asociado a un objeto, de forma que la exclusión mutua sólo se realiza entre el código sincronizado sobre el mismo objeto, y no entre código sincronizado en objetos distintos.
- Cuando se sincroniza un método de una clase, el objeto sobre el que se sincroniza es **this**.

```
class practica4 {  
  
    public synchronized void metodo1(){  
        // Esto se ejecuta en exclusión mutua  
    }  
    public synchronized void metodo2(){  
        // Esto se ejecuta en exclusión mutua  
    }  
}
```

Sólo es posible 1  
hilo en cualquiera  
de ellas

```
public static void main(String[] args) {  
  
    practica4 o1 = new practica4();  
    practica4 o2 = new practica4();  
  
    o1.metodo1();  
}
```

Pueden ser  
concurrentes

o2.metodo1();



# Sincronización en Java

## ■ EXCLUSIÓN MUTUA

- Es posible realizar la exclusión mutua de un trozo de código cualquiera, usando el bloque

```
synchronized(objeto) {  
      
}
```

- En este caso, el código encerrado en el bloque es mutuamente excluyente con cualquier otro código sincronizado con el mismo objeto.

```
class practica4 {  
    public synchronized void metodo1() {  
        // esto se ejecuta en exclusion mutua en this  
    }  
  
    public void metodo2() {  
        //esto puede ser concurrente  
        synchronized (this) {  
            // esto se ejecuta en exclusion mutua en this  
        }  
        //esto puede ser concurrente  
    }  
}
```

```
public static void main(String args[]) {  
    practica4 p4 = new practica4();  
    synchronized(p4){  
        //esto se ejecuta en exclusion mutua en p4  
    }  
}
```

Sólo es posible 1 hilo en cualquiera de ellas



# Sincronización en Java

## ▪ Sincronización

**Dentro del código sincronizado** es posible usar estas tres métodos (se heredan de `Object`) para controlar la ejecución de los hilos que lo ejecutan:

`.wait()`. Provoca la detención del hilo que hace la llamada. El hilo detenido libera la exclusión mutua, permitiendo a otros hilos acceder a la sincronización.

`.notify()`. Despierta uno de los hilos detenidos en `wait` del mismo objeto. Si no hay ningún hilo esperando en `wait`, no hace nada. El hilo que hace la llamada siempre continúa su ejecución, pero se garantiza la exclusión mutua dentro del código sincronizado

`.notifyAll()`. Similar a `notify`, pero despierta todos los hilos detenidos en `wait` del mismo objeto. Garantiza el acceso en exclusión mutua de los hilos despertados y el invocador, dentro del código sincronizado.

- Estos métodos deben ser llamados **dentro de un código sincronizado en el mismo objeto sobre que se invocan los métodos.**



# Sincronización en Java

- Sincronización

```
class practica4 {  
  
    public synchronized void metodo1() throws InterruptedException {  
  
        // esto se ejecuta en exclusion mutua en this  
        wait();  
  
        notifyAll();  
  
    }  
  
    public synchronized void metodo2() {  
  
        notify();  
  
    }  
  
    public void metodo3() throws InterruptedException {  
        //esto puede ser concurrente  
        synchronized (this) {  
            // esto se ejecuta en exclusion mutua en this  
            wait();  
        }  
        //esto puede ser concurrente  
    }  
  
    public static void main(String args[]) {  
  
        practica4 p4 = new practica4();  
  
        synchronized(p4){  
            //esto se ejecuta en exclusión mutua en p4  
            p4.notifyAll();  
        }  
    }  
}
```



# Sincronización en Java

- Sincronización

```
Object a = new Object();  
Object b = new Object();
```

```
public void metodo5()  
{  
  
    synchronized (a) {  
        a.wait();  
    }  
  
    synchronized (b) {  
        b.notify();  
    }  
}
```

```
public void metodo6()  
{  
  
    synchronized (a) {  
        a.notify();  
    }  
  
    synchronized (b) {  
        b.wait();  
    }  
}
```