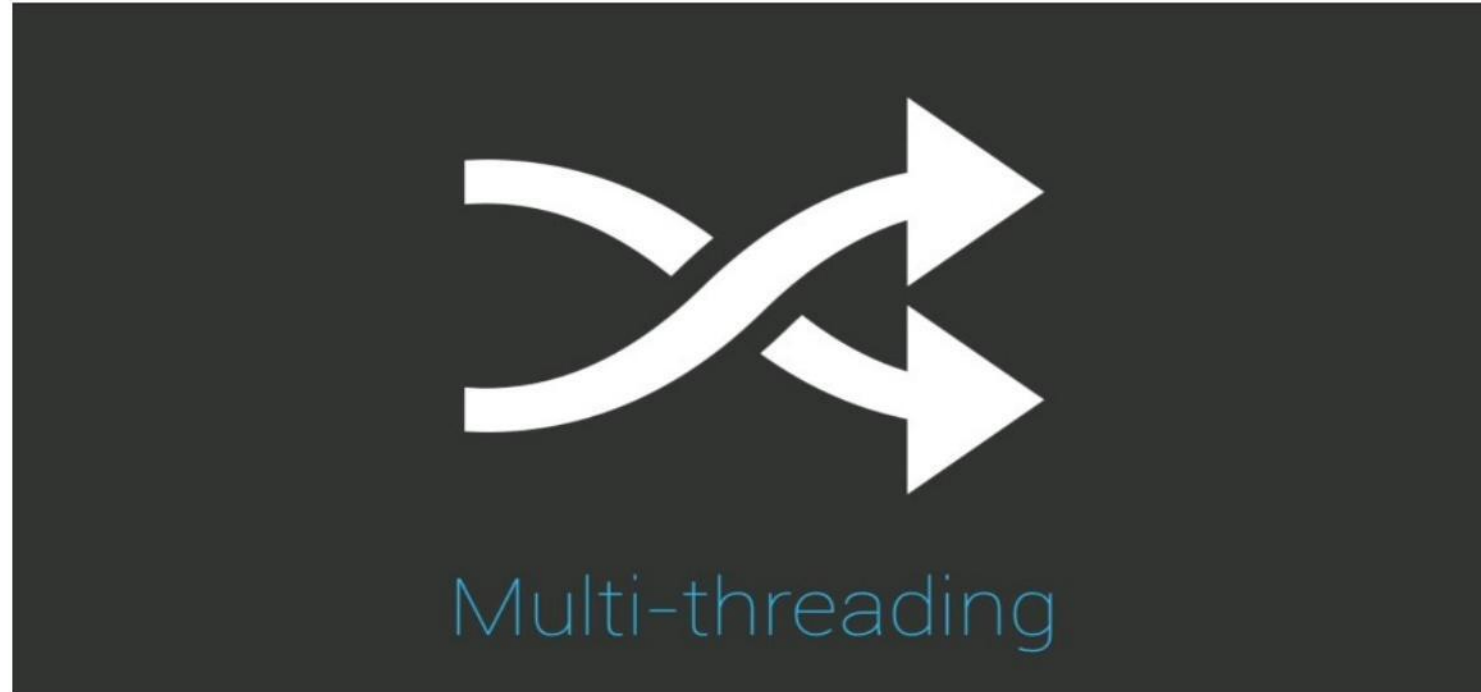
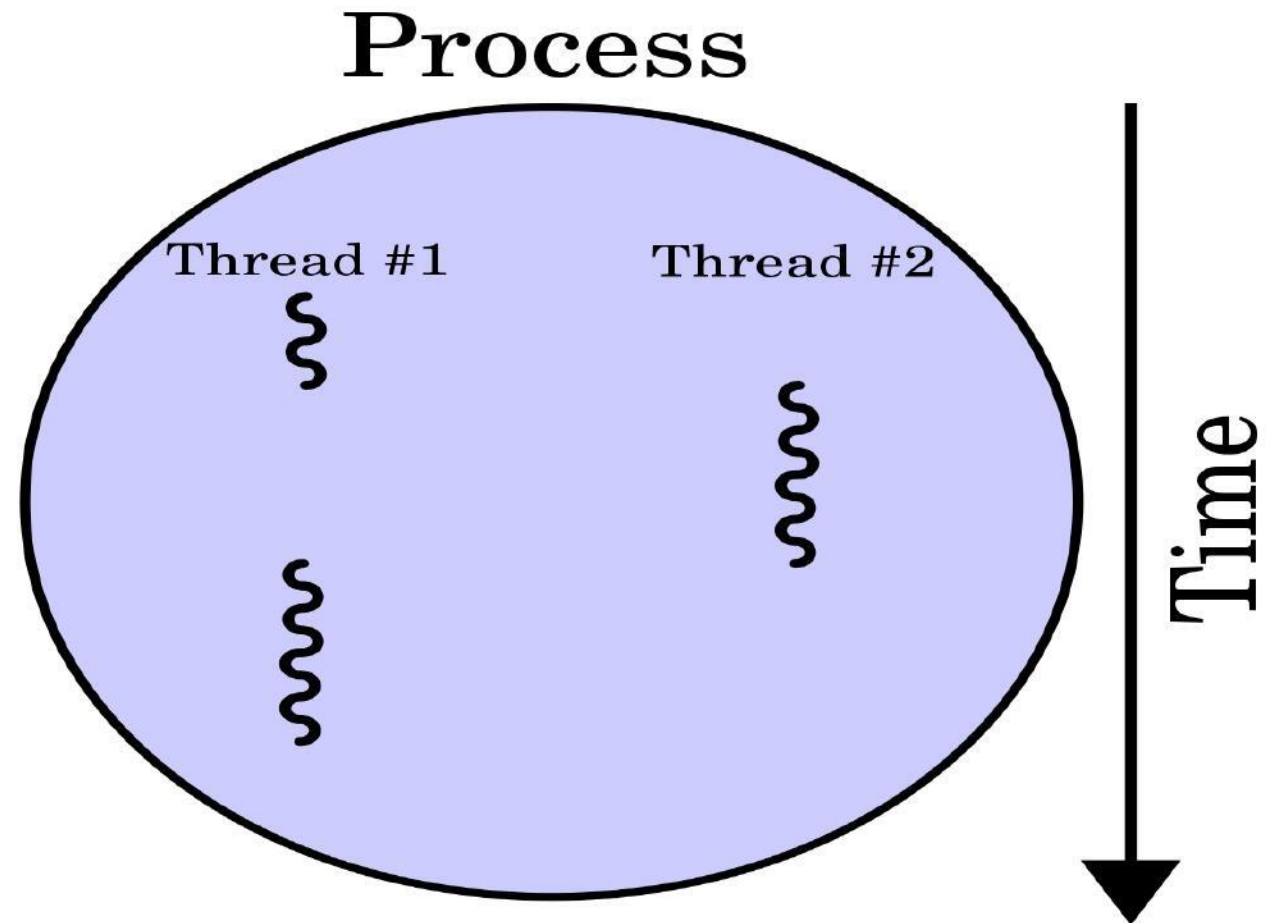


# UD02: Programació multifil



# 0. ÍNDEX

1. Classes per a la creació de fils
2. Gestió de fils
3. Gestió de prioritats
4. Sincronització entre fils

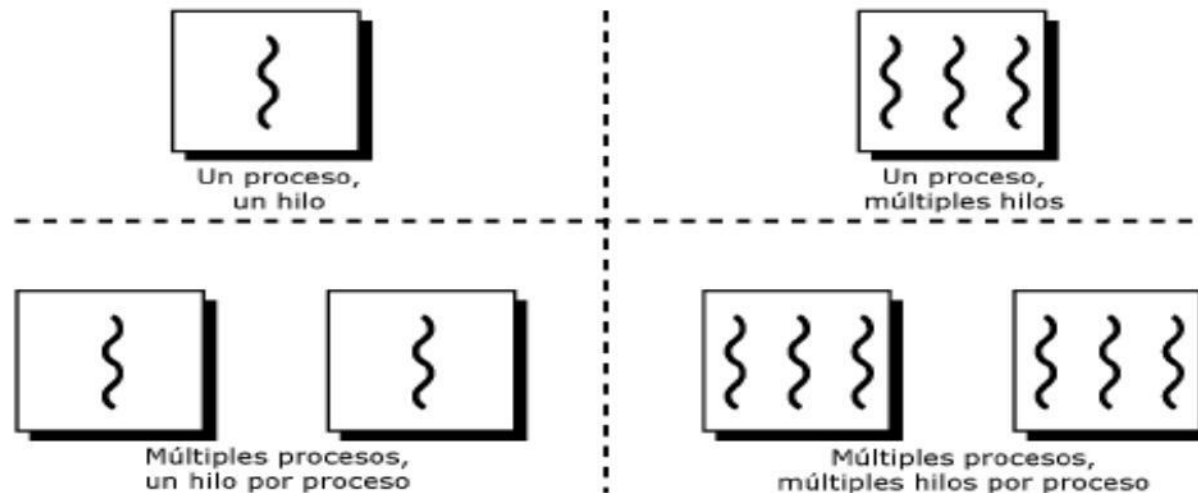


# 1. CLASSES PER A LA CREACIÓ DE FILS

## Definició

Un fil (també denominat “procés lleuger”) és una seqüència de codi en execució **dins del context d'un procés**. Els fils no poden executar-se ells sols, necessiten la supervisió d'un procés pare per a executar-se.

S'assemblen molt als processos ja que també comparteixen la CPU, només hi ha un fil actiu (en execució) en un instant donat i poden crear els seus propis fils fills. La principal diferència és que els fils **poden compartir recursos**, per la qual cosa quan un fil modifica una dada, els altres fils poden accedir a aqueixa dada modificada.



# 1. CLASSES PER A LA CREACIÓ DE FILS

## La classe Thread

A Java existeixen 2 formes per a crear fils: estenent la classe Thread o implementant la interfície Runnable. Ambdues són part del paquet java.lang.

Anem amb el primer mètode que consisteix a crear una subclasse de la classe Thread. Aquesta subclasse ha de sobreesciure el mètode **run()** amb les accions que el fil ha de desenvolupar. D'altra banda, la classe que vulga llançar aqueix mètode, el farà a través del mètode **start()**

```
public class Principal {  
  
    public static void main(String[] args) {  
        Hilo1 h = new Hilo1();  
        h.start();  
    }  
}
```

```
public class Hilo1 extends Thread {  
    public void run() {  
        System.out.println("Holi soy un hili");  
    }  
}
```





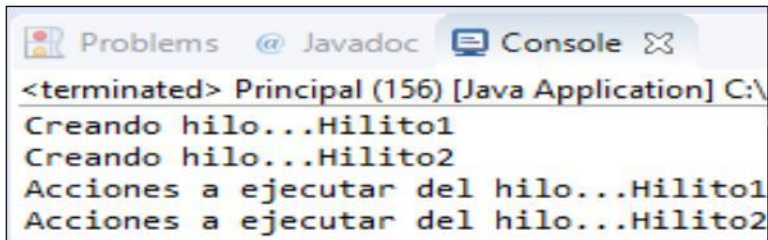
# 1. CLASSES PER A LA CREACIÓ DE FILS

## La classe Thread

A més, un fil pot tindre un nom i per a establir-li'l hem de passar-li'l des del constructor de la nostra classe a la classe Thread mitjançant la paraula reservada "super()". Més tard, ho podem recuperar amb el mètode "getName()":

```
public class Principal {  
  
    public static void main(String[] args) {  
        Hilo1 h = new Hilo1("Hilito1");  
        h.start();  
        Hilo1 h2 = new Hilo1("Hilito2");  
        h2.start();  
    }  
}
```

```
public class Hilo1 extends Thread {  
    public Hilo1(String nombre) {  
        super(nombre);  
        System.out.println("Creando hilo..." + getName());  
    }  
    public void run() {  
        System.out.println("Acciones a ejecutar del hilo..." + getName());  
    }  
}
```



```
<terminated> Principal (156) [Java Application] C:\  
Creando hilo...Hilito1  
Creando hilo...Hilito2  
Acciones a ejecutar del hilo...Hilito1  
Acciones a ejecutar del hilo...Hilito2
```

# 1. CLASSES PER A LA CREACIÓ DE FILS

MÈTODES	MISIÓ
sleep(mils)	Fa que el fil s'adorma durant "mils" mil·lisegons
getName()	Retorna el nom del fil
toString()	Retorna una representació en format cadena d'aquest fil incloent el nom, la prioritat i el grup de fils al qual pertany: Thread[Hilo1,2,main]
currentThread()	Retorna una referència a l'objecte fil actualment en execució (retorna per tant un objecte Thread)
activeCount()	Retorna el nombre de fils actius (tipus int)
getPriority	Retorna la prioritat del fil (tipus int)
setPriority(int p)	S'estableix la prioritat del fil que serà un enter que anirà d'1 a 10 i que per defecte és 5. Mínima: 1 Normal: 5 Màxima: 10

# 1. CLASSES PER A LA CREACIÓ DE FILS

## La classe ThreadGroup

S'utilitza per a manejar grups de fils en les aplicacions Java i estructurar millor el programa. La classe Thread proporciona constructors en els quals es pot especificar el grup del fil que s'està creant en el mateix moment d'instanciar-lo.

```
public class Principal {  
  
    public static void main(String[] args) {  
        ThreadGroup grupo = new ThreadGroup("Grupo de hilos");  
  
        GrupoHilos h = new GrupoHilos();  
        Thread h1 = new Thread(grupo,h,"Hilo1");  
        Thread h2 = new Thread(grupo,h,"Hilo2");  
        Thread h3 = new Thread(grupo,h,"Hilo3");  
  
        h1.start();  
        h2.start();  
        h3.start();  
  
    }  
}
```

```
public class GrupoHilos extends Thread{  
    public void run() {  
        System.out.println(Thread.currentThread().toString());  
    }  
}
```

Thread(grup, allò que executaran, nom)

```
Problems @ Javadoc Console X  
<terminated> Principal (156) [Java Application]  
Thread[Hilo1,5,Grupo de hilos]  
Thread[Hilo3,5,Grupo de hilos]  
Thread[Hilo2,5,Grupo de hilos]
```

# 1. CLASSES PER A LA CREACIÓ DE FILS

## Introducció als fils

Realitzarem un breu butlletí d'exercicis que ens permetrà posar en pràctica els conceptes vistos fins ara relacionats amb la gestió de fils i grups de fils en llenguatge Java.



UD02\_01\_Introducció\_fils



## 2. GESTIÓ DE FILS

### Estats d'un fil

**1-New:** quan es crea el fil però encara no s'executa.

**2-Runnable:** quan es diu al mètode **start()**, el fil prostateix a aquest estat.

**3-Dead:** quan finalitza el mètode **run()** el fil passa a estat dead encara que també és possible matar un fil intencionadament mitjançant l'ús de variables booleanes.



**DANGER**

Els mètodes **resume()**, **suspend()** i **stop()** estan en desús

**4-Bloquejat:** el fil es podria executar però hi ha alguna cosa que ho impedeix. El fil entra en aquest estat quan ocorre alguna de les següents accions:

4.1 Algú crida al **sleep()** del fil i el posa a dormir.

4.2 El fil s'està esperant a que es complete una operació de E/S.

4.3 El fil crida al mètode **wait()**. El fil no es tornarà a executar fins que reba els missatges **notify()** o **notifyAll()**.

4.4 El fil intenta bloquejar un objecte que està bloquejat per un altre fil.

**wait(), notify() i notifyAll() es voran més endavant en la part de sincronització**

## 2. GESTIÓ DE FILS

Ús d'una variable booleana per a parar un fil

```
public class Principal{  
    public static void main(String[] args) {  
        Hilo h = new Hilo();  
        h.start();  
        h.esperalseg();  
        h.pararHilo();  
    }  
}
```

```
public class Hilo extends Thread{  
    private boolean pararHilo = false;  
  
    public void run() {  
        while (!pararHilo) {  
            System.out.println("En el hilo!");  
        }  
    }  
  
    public void esperalseg() {  
        try {  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    public void pararHilo() {  
        pararHilo = true;  
        System.out.println("Hilo parado!");  
    }  
}
```



Problems @ Javadoc Console

<terminated> Principal (160) [Java Application]

En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
En el hilo!  
Hilo parado!

## 2. GESTIÓ DE FILS

### El mètode `getState()`

Aquest mètode ens retorna una constant que indica l'estat del fil els valors del qual són els següents:

Estat	Descripció
NEW	El fil encara no s'ha iniciat
RUNNABLE	El fil s'està executant
BLOCKED	El fil està bloquejat
WAITING	El fil crida a <code>wait()</code> i espera fins a rebre <code>notify()</code> o <code>notifyAll()</code>
TERMINATED	El fil ha finalitzat

Encara que **`wait()`** ho veurem en la part de sincronització, es diferencia amb `sleep()` en que **`wait()`** és de `Object` i **`sleep()`** és un mètode estàtic de `Thread`, però a més...

**`wait()` allibera el bloqueig mentre `sleep()` no allibera cap bloqueig mentre espera**



## 2. GESTIÓ DE FILS

### El mètode join()

El mètode join() provoca que el fil que fa la crida espere **la finalització** d'altres fils. Per exemple, si en el fil actual escric "h1.join()", el fil es queda en espera fins que mori el fil sobre el qual es realitza el join(), en aquest cas, el h1.

```
public class Principal {  
    public static void main(String[] args) {  
        Hilo h1 = new Hilo("Hilo1",3);  
        Hilo h2 = new Hilo("Hilo2",3);  
        h1.start();  
        h2.start();  
        try {  
            h1.join();  
            h2.join();  
        } catch (Exception e) {}  
        System.out.println("FIN DEL PROGRAMA")  
    }  
}
```

```
public class Hilo extends Thread {  
    private int n;  
    public Hilo(String nom, int n) {  
        super(nom);  
        this.n = n;  
    }  
  
    public void run() {  
        for (int i=1; i <= n; i++) {  
            System.out.println(getName() + ":" + i);  
            try {  
                sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
        System.out.println("Fin bucle " + getName());  
    }  
}
```



## 2. GESTIÓ DE FILS

### El mètode join()

Gràcies al mètode join() li diem al programa Principal que ha d'esperar-se al fet que finalitzen els fils h1 i h2 per a poder mostrar el missatge de "FI DE PROGRAMA", fixa't en la diferència entre utilitzar join i no utilitzar-ho.

#### AMB JOIN

```
<terminated> Principal (16)
Hilo1:1
Hilo2:1
Hilo2:2
Hilo1:2
Hilo1:3
Hilo2:3
Fin bucle Hilo2
Fin bucle Hilo1
FIN DEL PROGRAMA
```

Finalitzen en ordre indistint, ja que es llancen al mateix temps, però el missatge de FI sempre anirà al final si fem join.

#### SENSE JOIN

```
<terminated> Principal (16)
FIN DEL PROGRAMA
Hilo2:1
Hilo1:1
Hilo1:2
Hilo2:2
Hilo1:3
Hilo2:3
Fin bucle Hilo1
Fin bucle Hilo2
```

# 3. GESTIÓ DE PRIORITATS

## Definició

A l'hora de programar fils amb prioritats hem de tindre en compte que el comportament no està garantit i dependrà de la plataforma en la qual s'executen els programes i de les aplicacions que s'executen al mateix temps.

És a dir, és possible que un fil amb prioritat baixa s'execute abans que un fil amb prioritat alta, per la qual cosa no és un mecanisme del tot fiable. No obstant això, s'intenta garantir que **en la majoria de les ocasions** es done la situació inversa: és a dir, primer aquells amb **MAX\_PRIORITY** (10), després aquells amb **NORM\_PRIORITY**(5) i per a finalitzar, aquells amb **MIN\_PRIORITY**(1).

+getPriority(): retorna la prioritat del fil.  
+setPriority(): augmenta o disminueix la prioritat d'un fil:  
-h1.setPriority(Thread.MAX\_PRIORITY)  
-h1.setPriority(Thread.NORM\_PRIORITY)  
-h1.setPriority(Thread.MIN\_PRIORITY)

# 3. GESTIÓ DE PRIORITATS

## Introducció als fils

Realitzarem un breu butlletí d'exercicis que ens permetrà posar en pràctica els conceptes vistos fins ara relacionats amb la gestió de fils i gestió de prioritats en llenguatge Java.



UD02\_02\_Gestion\_fils



# 4. SINCRONITZACIÓ ENTRE FILS

## Mètodes sincronitzats

Els mètodes sincronitzats s'encarreguen de definir la “secció crítica”. La diem així per ser la secció de codi de l'objecte **compartit** a la qual intentaran accedir diversos fils al mateix temps, però que hem de sincronitzar perquè en aquesta zona **només hi haja 1 fil executant-se**.

Cada vegada que un fil intenta accedir a un mètode sincronitzat, pregunta si aquest ja està en possessió d'algun fil. Si és així, el fil s'espera i, quan el primer fil acaba la seua execució o se suspén, passa a executar-se el nou fil.

S'aconsegueixen mètodes sincronitzats afegint la paraula “synchronized” a la capçalera del mètode de l'objecte que es comparteix, quedant de la següent manera:

```
public class AumentarContador{  
    private int c = 0  
  
    public synchronized void incrementa(){  
        c = c +1;  
    }  
}
```



## 4. SINCRONITZACIÓ ENTRE FILS

### Blocs sincronitzats

Una versió una mica més simple que els mètodes sincronitzats són els blocs sincronitzats.

En aquesta ocasió, no és necessari definir un mètode sinó que en el mateix codi, fent ús de la paraula reservada “synchronized” seguida de l'objecte que s'està compartint entre parèntesi, duem a terme la mateixa operació.

```
synchronized (object){  
    //sentències crítiques  
}
```

Aqueix "object" serà compartit entre diversos fils d'execució, però gràcies a l'ús de synchronized garantim que només 1 fil execute alhora la secció crítica. És una solució més àgil que codificar un mètode, encara que impedeix la reutilització de codi.

## 4. SINCRONITZACIÓ ENTRE FILS

### Bloqueig de fils

Crearem una classe que defineix un mètode que rep un String i el pinta:

```
public class ObjetoCompartido {  
    public void PintaCadena(String s) {  
        System.out.print(s);  
    }  
}
```

D'altra banda, tindrem la classe "HiloCadena" que en el seu mètode run() invoca a PintaCadena() de la classe anterior:

```
public class HiloCadena extends Thread{  
    private ObjetoCompartido objeto;  
    String cad;  
  
    public HiloCadena(ObjetoCompartido c, String s) {  
        this.objeto = c;  
        this.cad = s;  
    }  
  
    public void run() {  
        for (int j=0; j < 10; j++)  
            objeto.PintaCadena(cad);  
    }  
}
```

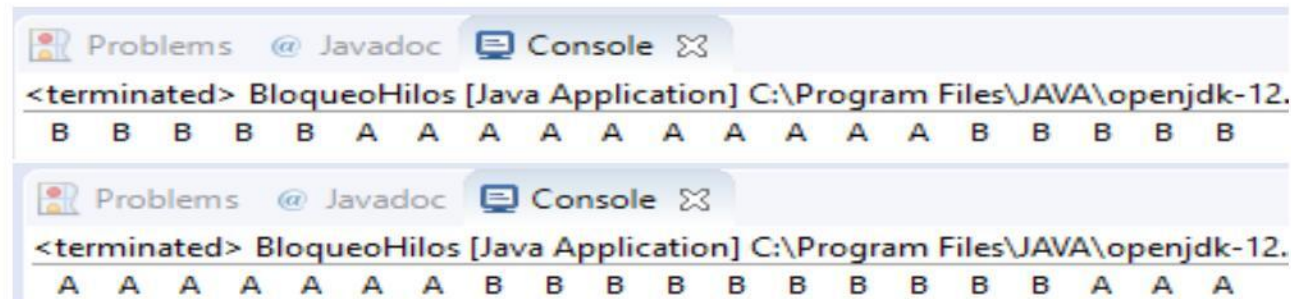
## 4. SINCRONITZACIÓ ENTRE FILS

### Bloqueig de fils

I finalment, el programa principal que crea l'objecte compartit i els 2 fils que el compartiran de manera sincronitzada:

```
public class BloqueoHilos {  
  
    public static void main(String[] args) {  
        ObjetoCompartido com = new ObjetoCompartido();  
        HiloCadena a = new HiloCadena(com, " A ");  
        HiloCadena b = new HiloCadena(com, " B ");  
        a.start();  
        b.start();  
    }  
}
```

Si llancem aquest programa principal, el resultat serà la impressió en pantalla de 10 As i de 10 Bs, l'ordre del qual variarà en cada execució...





## 4. SINCRONITZACIÓ ENTRE FILS

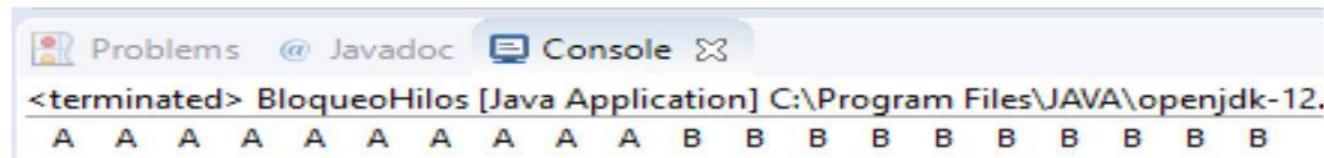
### Bloqueig de fils

I si volguérem traure alternativament A, B, A, B, etc...?, sembla que una manera podria ser sincronitzar aqueixa impressió dins del bucle. Ho deixem llavors així..

```
public void run() {  
    synchronized(objeto) {  
        for (int j=0; j < 10; j++)  
            objeto.PintaCadena(cad);  
    }  
}
```

Bloc sincronitzat que permet al fil A agafar la secció crítica, executar el codi, eixir i que després faça el mateix el fil B (o a l'inrevés, però mai al mateix temps)

No obstant això, el resultat \*por pantalla NO és l'esperat...



Això és pel fet que la sincronització evita que 2 crides del mateix objecte es mesclen, PERÒ NO GARANTEIX **L'ORDRE DE LES CRIDES**.



## 4. SINCRONITZACIÓ ENTRE FILS

### Bloqueig de fils

Si el que volem és que, a més de que les crides no es mesclen, puguem garantir l'ORDRE **DE LES CRIDES**, és necessari mantindre una certa coordinació entre els 2 fils, és a dir, que els fils “es parlen i s'organitzen”. A aquest efecte utilitzarem els mètodes **wait()**, **notify()** i **notifyAll()**.

Nom del mètode	Descripció
Objecte.wait()	Un fil que crida al mètode “wait()” d'un cert objecte queda suspès fins que un altre fil crida al mètode “notify()” o “notifyAll()” del mateix objecte. En quedar-se esperant, “obri” la SC perquè altres fils la puguin executar, d'una altra forma existiria un bloqueig.
Objecte.notify()	Desperta només a un dels fils que va realitzar una crida a “wait()” sobre el mateix objecte. L'elecció és arbitrària.
Objecte.notifyAll()	Desperta a tots els fils que estan esperant l'objecte. És una generalització del mètode anterior.

## 4. SINCRONITZACIÓ ENTRE FILS

### Bloqueig de fils

```
public void run() {  
    synchronized(objeto) {  
        for (int j=0; j < 10; j++) {  
            objeto.PintaCadena(cad);  
            objeto.notify(); //libera un hilo esperando  
            try {  
                objeto.wait(); //esperar al notify. Abre la SC  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
    //Despertamos a todos los waits sobre el objeto  
    objeto.notifyAll();  
} //fin bloque synchronized
```

synchronized = només 1 fil  
executant-se en la SC

Problems @ Javadoc Console

<terminated> BloqueoHilos [Java Application] C:\Program Files\JAVA\openjdk-12.

A B A B A B A B A B A B A B A B A B A B

### FUNCIONAMENT:

+1-Entra el hiloA, imprimeix la A, crida a notify() i no hi ha ningú esperant així que no desperta a ningú. Després crida a wait() i es queda esperant, obrint la SC.

+2-Com el hiloA va obrir la SC, entra el hiloB, que imprimeix la B, crida a notify() despertant al HiloA i crida a wait(), esperant i tornant a obrir la SC.

+3-El fil A torna a executar-se imprimint la A...

+4-El fil A i el fil B alternen la CPU fins que arriben a 10.

+5-Amb notifyAll() es desperten tots els waits perquè l'aplicació finalitzi correctament.

# 4. SINCRONITZACIÓ ENTRE FILS

## Bloqueig de fils

Realitzarem un breu butlletí d'exercicis que ens permetrà posar en pràctica els conceptes vistos fins ara relacionats amb la sincronització de fils en llenguatge Java.



UD02\_03\_Sincronitzacio\_fils