Ejemplo de Concurrencia - Indeterminismo - En Java

Consideremos el trozo de código siguiente:

```
public class Contador {
      public static void main(String[] args) throws InterruptedException {

            class Cdor {
                  private int valor = 0;
                  public  void incrementar(){ ++valor;}
                  public int getValor() {return valor;}
            }

            final Cdor unContador = new Cdor();

            class HiloCdor extends Thread {
                  public void run() {
                        for (int x=0; x<10000; ++x)
                              unContador.incrementar();
                  }
            }

            Hilo1Cdor h1= new HiloCdor();
            Hilo2Cdor h2= new HiloCdor();
            h1.start();
            h2.start();
            h1.join();
            h2.join();
            System.out.println("en main-" + unContador.getValor());
      }

}
```

Este ejemplo utiliza clases internas (inner classes) de Java. Una clase interna está asociada con una instancia de la clase que la engloba, y tiene acceso directo a los métodos y variables. Una instancia de una clase interna solo puede existir dentro de una instancia de la clase externa. El uso de clases internas es una forma de lograr agrupar clases que solamente son utilizadas en un lugar.
En el caso del ejemplo utilizamos clases internas dado que el objetivo es mostrar la necesidad de sincronización entre 2 hilos de ejecución que comparten una variable, (no realizar una clase que queramos reutilizar).

Se considera una ***clase interna Cdor***, y una ***clase interna HiloCdor*** que es subclase de Thread y cuyo método ***run()*** actúa sobre una instancia de la clase Cdor. Ambas, al ser clases internas, sólo existen dentro de la clase Contador, que es la clase dentro de la cual están definidas.

Se crea una instancia de Cdor, y 2 instancias de HiloCdor, es decir que se tienen 2 hilos de ejecución que se ejecutan de manera concurrente y actúan sobre la misma instancia unContador. El objeto referenciado por unContador es una variable compartida.

Al ejecutar el código anterior repetidas veces se obtienen resultados diferentes, cercanos al 20000, pero en general NO el 20000 esperado. Esto se debe a que ambos hilos ejcutan el método *run*(), que repetidamente intenta *incrementar* el valor de unContador. La acción de incrementar (++valor) no es atómica, es decir consiste de recuperar el valor, agregarle 1 y actualizar el valor, es equivalente a hacer valor:= valor +1. Por esta razón es posible que cuando le toque el turno de ejecución al hilo h2, el hilo h1 sea interrumpido justo después de haber recuperado el valor pero antes de modificarlo, y asi se pierdan algunos incrementos.

Recordemos que al comenzar a ejecutar un progama en Java, se activa se crea el hilo de ejecución principal, que se ejecuta de manera concurrente con los otros hilos creados por el programa. En esta situación el efecto de la instrucción *h1.join()* es lograr que el hilo h1 *(y el hilo h2)* termine su ejecución antes de que el hilo principal pueda continuar.
Asi la instrucción

```
System.out.println("en main-" + unContador.getValor());
```

muestra el valor final de unContador.

Si se comentan las líneas correspondientes a mensaje join() enviado a h1 y h2, entonces el mensaje del main se mostrará un valor que no es el final.
A los efectos de ver bien como va avanzando la ejecución de los hilos se propone agregar mensajes de muestra.

Para poder hacer un mejor seguimiento de la ejecución entralazada de los hilos considerar el siguiente código que tiene algunas modificaciones con respecto al codigo anterior, pero permite agregar carteles apropiados para poder analizarlo.

```
public class Contador {
   public static void main(String[] args) throws InterruptedException {

      class Cdor {
            private int valor = 0;
            public  void incrementar(){ ++valor;
                  System.out.println(valor);}
            public int getValor() {return valor;}
      }

      final Cdor contador = new Cdor();

      class Hilo1Cdor extends Thread {
            public void run() {
                  for (int x=0; x<9000; ++x){
                        System.out.print("x-" + x + "-" );
                        contador.incrementar();
                  }
                  System.out.println("en hilo1-" + contador.getValor());
            }
      }


      class Hilo2Cdor extends Thread {
            public void run() {
                  for (int y =0; y<8000; ++y){
                        System.out.print("y-" + y + "-" );
                        contador.incrementar();
                        }
                  System.out.println("en hilo2-" + contador.getValor());
            }
      }

      Hilo1Cdor h1= new Hilo1Cdor();
      Hilo2Cdor h2= new Hilo2Cdor();
      h1.start(); h2.start();
      h1.join();  h2.join();
      System.out.println("en main-" + contador.getValor());
   }

}
```

Se puede realizar cambiando las clases internas y considerando la clase Datos, y una clase para los hilos

```
public class Contador {
	public static void main(String[] args) throws InterruptedException {

		Datos contador = new Datos(0);
		ProcesoUno p1= new ProcesoUno(contador);
		ProcesoUno p2= new ProcesoUno(contador);
		Thread h1= new Thread(p1);
		Thread h2= new Thread(p2);
		h1.start(); h2.start();
		h1.join();  h2.join();
		System.out.println("en main-" + contador.getDato());
	}
}
```

```
public class Datos {
	private int dato;

	public Datos(int nro){
		dato = nro;
	}

	public int getDato(){
		return dato;
	}

	public void incrementar(){
		dato++;
	}
}
```

```
public class ProcesoUno implements Runnable{
	private Datos unDato;

	public ProcesoUno(Datos unD){
		unDato = unD;
	}

	public void run(){
		for (int i=1; i<10000; i++){
			unDato.incrementar();
		}
	}
}
```

Entonces ... es necesario sincronizar (es decir coordinar) el trabajo sobre la variable compartida, en particular se debe sincronizar el método incrementar(), que es el que modifica el valor de la variable. Se utiliza la palabra reservada "synchronized" de la forma siguiente:

```
	class Cdor {
		private int valor = 0;
		public  synchronized void incrementar(){ ++valor;}
		public int getValor() {return valor;}
	}
```

O en la clase datos

```
public class Datos {
	private int dato;

	public Datos(int nro){
		dato = nro;
	}

	public int getDato(){
		return dato;
	}

	public synchronized void incrementar(){
		dato++;
	}
}
```