2.7. COMUNICACION Y SINCRONIZACION DE HILOS

A menudo los hilos necesitan comunicarse unos con otros, la forma de comunicarse consiste usualmente en compartir un objeto. En el siguiente ejemplo dos hilos comparten un objeto de la clase Contador. Esta clase define un atributo contador y tres métodos, uno de ellos incrementa una unidad su valor, el otro lo decrementa y el tercero devuelve su valor; el constructor asigna un valor inicial al contador:

```
class  Contador {

private  int  c    = 0;      //atributo  contador

Contador(int  c){  this.c=c;        }

public  void  incrementa()        { c =  c + 1; }

public  void  decrementa()        { c =  c -1; }

public  int  valor(){ return  c; }

}//  CONTADOR
```

Para probar el objeto compartido se definen dos clases que extienden Thread. En la clase HiloA se usa el método del objeto contador que incrementa en uno su valor. En la clase HiloB se usa el método que decrementa su valor. Se añade un sleep() intencionadamente para probar que un hilo se duerma y mientras el otro haga otra operación con el contador, así la CPU no realiza de una sola vez todo un hilo y después otro y podemos observar mejor el efecto:

```
class  HiloA extends  Thread {
        private  Contador  contador;
                public  HiloA(String  n,   Contador  c) {
                setName(n);
                contador  =  c;
        }
public  void run()  {
for (int  j = 0; j <300;  j++) {
        contador.incrementa(); //incrementa  el  contador
        try{
                sleep(100);
                } catch  (InterruptedException  e)         {}
                }
                System.out.println(getName() + "  contador  vale  " + contador.valor());
```

```
                }
 }
)// FIN HILOA


class HiloB extends Thread {
        private Contador contador;
        public HiloB(String n, Contador c)    {
        setName(n);
        contador = c;
}
        public void run(){
                for(int j=0; j <300; j++)
                        contador.decrementa();          //decrementa el contador
                        try     {
                                sleep(100);
                        catch   (InterruptedException e)        {}
}
                System.out.println(getName()  "  contador vale" +contador. valor());
}
}// FIN HILOB
```

A continuación, se crea el método main(), donde primero se define un objeto de la clase Contador y se le asigna el valor inicial de 100. A continuación, se crean los dos hilos pasándoles dos parámetros: un nombre y el objeto Contador. Seguidamente se inicia la ejecución de los hilos:

```
public class CompartirInf1{     {
                public static void main(String[] args)  {
                        Contador cont = new Contador(100);
                        HiloA a= new HiloA("HiloA", cont);
                        HiloB b - new HiloB("HiloB", cont);
                        a.start();
                        b.start();
```

```
            }
        }
```

Nos puede dar la impresión que al ejecutar los hilos el valor del contador en el hilo A debería ser 400, ya que empieza en 100 y le suma 300; y en B, 100 ya que se resta 300; pero no es así. Al ejecutar el programa los valores de salida pueden no ser los esperados y variará de una ejecución a otra:

HiloB contador vale 100

HiloA contador vale 100

Al probarlo sin el método sleep() da la sensación de que la salida es la esperada, pero no siempre nos va a dar dicha salida:

HiloA contador vale 400

HiloB contador vale 100

HiloB contador vale 100

HiloA contador vale 100

HiloA contador vale 43

HiloB contador vale 43

HiloB contador vale 100

HiloA contador vale 400

HiloB contador vale -200

HiloA contador vale 100

2.7.1. BLOQUES SINCRONIZADOS

Una forma de evitar que esto suceda es hacer que las operaciones de incremento y decremento del objeto contador se hagan de forma atómica, es decir, si estamos realizando la suma nos aseguramos de que nadie realice la resta hasta que no terminemos la suma. Esto se puede lograr añadiendo la palabra **synchronized** a la parte de código que queramos que se ejecute de forma atómica. Java utiliza los bloques synchronized para implementar las regiones críticas. El formato es el siguiente:

```
synchronized    (object){
```

```
//sentencias críticas
}
```

Los métodos run() de las clases HiloA e HiloB se pueden sustituir por los siguientes; para el HiloB:

```
public void  run() {
        synchronized (contador) {
         for (int  j = 0;  j < 300;  j++)       {
                contador.decrementa();
        }
        System.out.println(getName()   +  "  contador vale  "+  contador.valor());
        }
}
```

Para el HiloA:

```
public void run(){
        synchronized    (contador)       (
        for (int  j=0;   j<300;  j++) {
                contador.incrementa();
        }
        System.out.println(getName()   +  "  contador  vale  "+  contador.valor());
        }
}
```

El bloque synchronized o región crítica (que aparece sombreado) lleva entre paréntesis la referencia al objeto compartido Contador. Cada vez que un hilo intenta acceder a un bloque sincronizado le pregunta a ese objeto si no hay algún otro hilo que ya le tenga bloqueado. Es decir, le pregunta si no hay otro hilo ejecutando algún bloque sincronizado con ese objeto. Si está tomado por otro hilo, entonces el hilo actual se suspende y se pone en espera hasta que se libere el bloqueo. Si esta libre, el hilo actual bloquea el objeto y ejecuta el bloque; el siguiente hilo que intente ejecutar un bloque sincronizado con ese objeto, será puesto en espera. El bloqueo del objeto se libera cuando el hilo que lo tiene tornado sale del bloque porque termina la ejecución, ejecuta un return o lanza una excepción.

```
HiloA contador  vale      400
HiloB  contador vale      100
```

Si se cambia el orden de la ejecuciOn de los hilos, primero el HiloB y luego el HiloA, la salida será:

HiloB contador vale -200

HiloA contador vale 100

ACTIVIDAD 2.7.1

Crea un programa Java que lance cinco hilos, cada uno incrementara una variable contador de tipo entero, compartida por todos, 5000 veces y luego saldrá. Comprobar el resultado final de la variable. ¿Se obtiene el resultado correcto? Ahora sincroniza el acceso a dicha variable. Lanza los hilos primero mediante la clase Thread y luego mediante el interfaz Runnable. Comprueba el resultado.