ut2. Programación multihilo.

Métodos sincronizados

Forma comunicación hilos

Necesidad comunicación entre hilos.

Solución → Compartir un objeto.

Planteamos ejemplo en el que dos hilos compartirán un objeto de la clase Contador.

Clase contador

```
class Contador {

        private int c=0; //atributo contador

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

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

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

}//CONTADOR
```

La clase define un atributo contador.

Tenemos tres métodos incrementa (aumenta valor en 1), decrementa (decrementa valor en 1) y getValor (devuelve el valor).

El constructor asigna valor inicial a contador.

HiloA

```
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.getValor());
                }
}//FIN HILOA
```

HiloB

```
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.getValor());
                }

}//FIN HILOB
```

Método principal (main)

```
public class CompartirInf1{
        public static void main (String[] args){
                Contador cont=new Contador(100); //creo contador valor inicial 100
                HiloA a=new HiloA("HiloA",cont); //creo hiloA (incrementa) y le paso contador
                HiloB b=new HiloB("HiloB",cont); //creo hiloB (deceremenTa) y le paso contador
                a.start(); //inicio hiloA
                b.start(); //inicio hiloB

        }


}
```

Ejecutamos

```
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ javac contador.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ javac HiloA
HiloA2.java  HiloA.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ javac HiloA.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ javac HiloB.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ javac CompartirInf1.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos $ java CompartirInf1
```

El sistema no devuelve nada. Debemos sincronizar el acceso a objetos comunes.

descargar ejemplo

Acceso Objetos comunes

Operaciones sobre objetos comunes (incremento y decremento) deben realizarse de forma atómica (hasta que no termine la suma no se realiza la resta).

¿Cómo lo logramos?

→ Añadimos **Synchronized** a la parte de código que queremos se ejecute de forma atómica.

Synchronized

```
sinchronized (object) {
    //sentencias críticas
}
```

Cada vez que un hilo intenta acceder a un bloque sincronizado le pregunta a ese objeto si no hay algún otro hilo que ya lo tenga bloqueado. Si está ocupado el hilo se suspende, si está libre lo tiene tomado hasta que termina ejecución

HiloA sinchronized

```
class HiloA extends Thread{
        private Contador contador;

        public HiloA(String n,Contador c){
                setName(n);
                contador=c;
        }

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

Modificamos HiloA. Añadimos synchronized y eliminamos captura interrupción.

HiloB Sinchronized

```
class HiloB extends Thread{
        private Contador contador;

        public HiloB(String n,Contador c){
                setName(n);
                contador=c;
        }

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

}//FIN HILOB
```

Modificamos run → Añadimos synchronized y eliminamos captura interrupción

Ejecutamos tras las modificaciones

```
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/sinchronized $ javac HiloA.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/sinchronized $ javac HiloB.java
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/sinchronized $ javac CompartirInf1.java

david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/sinchronized $ java CompartirInf1
HiloA contador vale 400
HiloB contador vale 100
```

descargar ejemplo

Práctica Cuenta1

Completa la Clase Cuenta que se facilita para que en ella se defina un atributo saldo y tres métodos, uno devuelve el valor del saldo, otro resta el sado una cantidad y el tercero realiza las comprobaciones para hacer la retirada del dinero. El constructor inicial el saldo actual.

Clase Cuenta

Declaración clase y constructor

```
class Cuenta{
        private int saldo;

        //constructor
        Cuenta (int s)
        {
                saldo=s; //inicializa el saldo actual
        }//-------------------------------------
```

Clase Cuenta

Método RetirarDinero

```
//comprueba se pueda retirar dinero  y lo retira
void RetirarDinero(int cant,String nom)
{
        if (getSaldo()>=cant){
                System.out.println ("Se va a retirar saldo (actual es: "+ getSaldo() +")");
                try {
                        Thread.sleep(500);
                        } catch (InterruptedException ex) {}
                restar (cant);
                System.out.println (nom +" retira => : "+cant +". Saldo Actual: "+ getSaldo() +"");
        }
        else{
                System.out.println (nom +" No puede retirar dinero no hay saldo. Saldo Actual: "+ getSaldo() +"");
        }

}// fin retirar dinero-----------------------------
```

Clase Cuenta. Métodos a desarrollar

```
//devuelve el saldo
int getSaldo()
{

}//--------------------------------

// resta la cantidad al saldo
void restar(int cantidad)
{
        |
}//-----------------------------------
```

Clase SacarDinero

Creamos clase hilo que utilizará objeto cuenta. Simulamos acceso común a cuenta mediante clase creará hilos accederán a objeto cuenta y compartirán información.

SacarDinero

```
class SacarDinero extends Thread{
        private Cuenta c; //declaro objeto cuenta c
        String nom;

        //constructor
        public SacarDinero (String n, Cuenta c){
                super(n);
                this.c=c;
        }
        //run
        public void run(){
                for (int x=1;x<=4;x++){
                        c.RetirarDinero(10,getName());
                }
        }

}
```

Usa clase cuenta para retirar el dinero.

run realiza bucle retira dinero 10, y obtiene el nombre del hilo

Método Compartir Información

```
public class CompartirInf3{
        public static void main(String[] args){
                Cuenta c=new Cuenta(40);
                SacarDinero h1=new SacarDinero ("Ana",c);
                SacarDinero h2=new SacarDinero ("Juan",c);

                h1.start();
                h2.start();
        }


}
```

Ejecución

```
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/Cuenta $ java CompartirInf3
Se va a retirar saldo (actual es: 40)
Se va a retirar saldo (actual es: 40)
Juan retira => : 10. Saldo Actual: 30
Se va a retirar saldo (actual es: 30)
Ana retira => : 10. Saldo Actual: 30
Se va a retirar saldo (actual es: 30)
Juan retira => : 10. Saldo Actual: 20
Se va a retirar saldo (actual es: 20)
Ana retira => : 10. Saldo Actual: 10
Se va a retirar saldo (actual es: 10)
Juan retira => : 10. Saldo Actual: 0
Juan No puede retirar dinero no hay saldo. Saldo Actual: -10
Ana retira => : 10. Saldo Actual: -10
Ana No puede retirar dinero no hay saldo. Saldo Actual: -10
```

Observa que permite retirar saldo cuando este era 0.

Desarrollo. Práctica cuenta

Realiza modificaciones en las clases para que el método adecuado se ejecute de forma atómica e indivisible.

Para aplicar synchronized a un método determinado:

```
synchronized public void metodo(){
    }
```

Desarrollo

Ejecución tras modificaciones debe ser:

```
david@david-OEM ~/pss/ut2/3 Comunicación y sincronización de hilos/Práctica Cuenta $ java CompartirInf3
Se va a retirar saldo (actual es: 40)
Ana retira => : 10. Saldo Actual: 30
Se va a retirar saldo (actual es: 30)
Juan retira => : 10. Saldo Actual: 20
Se va a retirar saldo (actual es: 20)
Ana retira => : 10. Saldo Actual: 10
Se va a retirar saldo (actual es: 10)
Juan retira => : 10. Saldo Actual: 0
Ana No puede retirar dinero no hay saldo. Saldo Actual: 0
Juan No puede retirar dinero no hay saldo. Saldo Actual: 0
Ana No puede retirar dinero no hay saldo. Saldo Actual: 0
Juan No puede retirar dinero no hay saldo. Saldo Actual: 0
```