**ACTIVIDAD 15**

**Enunciado**

Realizar en Java un programa que cree 3 hilos y que su clase principal se llame Actividad15. Cada uno de ellos mostrará 5 mensajes por pantalla indicando su nombre y la hora actual. Entre mensaje y mensaje de un mismo hilo se esperará 1 segundo.

El resultado esperado sería equivalente al siguiente, teniendo en cuenta que el orden de aparición de los 5 mensajes de cada hilo puede variar de una ejecución a otra del programa y que la hora actual de ejecución se corresponderá con la hora en la que se ejecute dicho programa:

Hilo 1 - 18:54:03

Hilo 3 - 18:54:03

Hilo 2 - 18:54:03

Hilo 1 - 18:54:04

Hilo 3 - 18:54:04

Hilo 2 - 18:54:04

Hilo 1 - 18:54:05

Hilo 3 - 18:54:05

Hilo 2 - 18:54:05

Hilo 3 - 18:54:06

Hilo 1 - 18:54:06

Hilo 2 - 18:54:06

Hilo 3 - 18:54:07

Hilo 1 - 18:54:07

Hilo 2 - 18:54:07

**Solución**

**package** ejer15;

**public** **class** Actividad15 {

**public** **static** **void** main(String[] args) {

// Crear tres hilos

MiHilo hilo1 = **new** MiHilo("Hilo 1");

MiHilo hilo2 = **new** MiHilo("Hilo 2");

MiHilo hilo3 = **new** MiHilo("Hilo 3");

// Iniciar los hilos

hilo1.start();

hilo2.start();

hilo3.start();

}

}

---------------------------------------------

**package** ejer15;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**class** MiHilo **extends** Thread {

**private** String nombreHilo;

**public** MiHilo(String nombreHilo) {

**this**.nombreHilo = nombreHilo;

}

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

// Obtener la hora actual

String horaActual = **new** SimpleDateFormat("HH:mm:ss").format(**new** Date());

// Imprimir el mensaje

System.***out***.println(nombreHilo + " - " + horaActual);

// Esperar 1 segundo

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

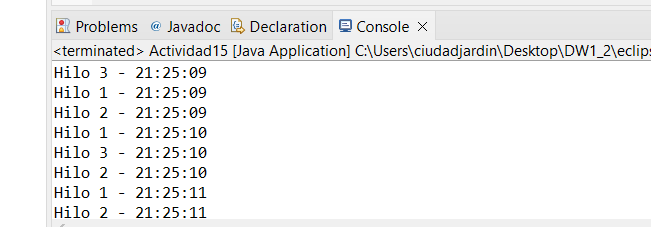
e.printStackTrace();

}

}

}

}



**Resolución de problemas**

1. **No imprime nada:**
   * **Solución:** Usa **hilo.start()** en lugar de **hilo.run()**.
2. **Excepciones de interrupción:**
   * **Solución:** Maneja **InterruptedException** en **sleep()**.
3. **Salida no intercalada:**
   * **Solución:** Comportamiento normal en hilos; no requiere sincronización.
4. **Hora incorrecta:**
   * **Solución:** Usa **SimpleDateFormat("HH:mm:ss")**.
5. **Programa se cierra antes de terminar:**
   * **Solución:** Usa **hilo.join()** para esperar a que terminen los hilos.

**ACTIVIDAD 16**

**Enunciado**

Modificar el siguiente programa en Java para conseguir el siguiente resultado:

HiloA contador vale 400

HiloB contador vale 100

La clase principal de este programa se llamará Actividad16 y la modificación a realizar requerirá sincronizar el objeto contador, de tal forma que el HiloA incremente 300 veces el contador, bloqueando el objeto, para que posteriormente el HiloB lo decremente.

El detalle del programa a modificar es el siguiente:

**public** **class** Actividad16 {

**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();

}

}

**class** Contador {

**private** **int** c=0;

Contador (**int** c) {

**this**.c = c;

}

**public** **void** incrementa() {

c=c+1;

}

**public** **void** decrementa() {

c=c-1;

}

**public** **int** getValor() {

**return** c;

}

}

**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();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {}

}

System.***out***.println(getName() + " contador vale" + contador.getValor());

}

}

**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();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {}

}

System.***out***.println(getName() + " contador vale" + contador.getValor());

}

}

**Solución**

**package** ejer16;

**public** **class** Actividad16 {

**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();

**try** {

a.join(); // Esperar a que HiloA termine antes de iniciar HiloB

} **catch** (InterruptedException e) {

e.printStackTrace();

}

b.start();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** ejer16;

**class** Contador {

**private** **int** c = 0;

Contador(**int** c) {

**this**.c = c;

}

**public** **synchronized** **void** incrementa() {

c = c + 1;

}

**public** **synchronized** **void** decrementa() {

c = c - 1;

}

**public** **synchronized** **int** getValor() {

**return** c;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** ejer16;

**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();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.println(getName() + " contador vale " + contador.getValor());

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** ejer16;

**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 < 100; j++) { // Decrementar 100 veces

contador.decrementa();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

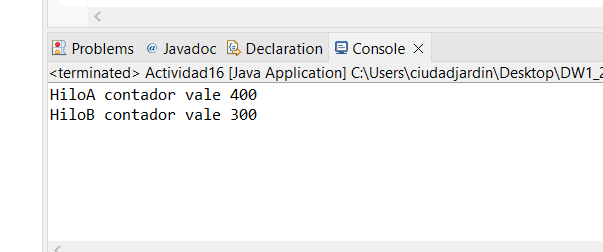
}

}

System.***out***.println(getName() + " contador vale " + contador.getValor());

}

}



**Resolución de problemas**

1. **Sincronización:** Asegúrate de que los métodos de acceso al contador estén sincronizados para evitar condiciones de carrera.
2. **Orden de Ejecución:** Usa **join()** para controlar el orden de ejecución de los hilos.
3. **Excepciones:** Maneja adecuadamente las excepciones de interrupción.

**REQUISITOS**

1. Entregar las soluciones de las actividades en un único archivo comprimido cuyo nombre es: el nombre original de la actividad, seguido de un guión bajo y del primer apellido del alumno. Donde tendremos:
   1. Un archivo de extensión .odt o .docx que contendrá
      1. Una copia del código del programa
      2. Una captura de pantalla del resultado de la ejecución del programa
   2. Los archivos con el código fuente de extensión .java

**CRITERIOS DE CORRECCIÓN**

1. Cumplimiento de los requisitos (15%).
2. Hacer lo que se indica en el enunciado (55%).
3. Claridad del código Java (15%).
4. Documentación del código Java (15%).
5. Explicación detallada de cómo se han resuelto problemas que hayan podido surgir en la realización de la actividad (hasta un 15% adicional sobre la nota obtenida). Dicha explicación se incluirá al principio de los archivos de código fuente Java mediante líneas de comentarios. La primera línea de comentarios sólo incluirá lo siguiente: // Resolución de problemas.
6. Variables no usadas (-15%).
7. Librerías no usadas (-15%).