UT-2: Programación de hilos

Ejercicios de refuerzo y afianzamiento de conceptos estudiados.

1. Creación de hilos.
    a. Crea una clase, **MiHilo1**, que cree un hilo mediante la implementación la interface: **Runnable** y en su método **run** muestre un mensaje por pantalla. Ejercicio que crea una clase para la creación de hilos mediante Runnable.

    b. Crea una clase, **MiHilo2**, que cree un hilo mediante la herencia de: **Thread** y en su método **run** muestre un mensaje por pantalla. Ejercicio que crea una clase para la creación de hilos mediante Thread.

    (Ver 2.1).

    c. Amplia el ejercicio: **1-a** añadiendo un constructor que reciba un parámetro **id** que será el número de hilo. Modifica el nombre a: **MiHilo3**. Añade un atributo **prívate** de tipo **int** y nombre **id** que tomará valor en el constructor. Modifica el mensaje a mostrar para que aparezca el valor del hilo. Ejercicio que crea una clase para la creación de hilos mediante Runnable con un parámetro de número de hilo.

d. Amplia el ejercicio: **1-b** añadiendo un constructor que reciba un parámetro **id** que será el número de hilo. Modifica el nombre a: **MiHilo4**. Añade un atributo **prívate** de tipo **int** y nombre **id** que tomará valor en el constructor. Modifica el mensaje a mostrar para que aparezca el valor del hilo. Ejercicio que crea una clase para la creación de hilos mediante Thread con un parámetro de número de hilo.

(Ver 2.2).

e. Modifica el **1-c** para que el hilo creado con **Runnable** se comporte como un hilo creado con Thread aún implementando a Runnable. Modifica el nombre a: **MiHilo5**. Modifica el mensaje para indicar esta nueva característica. Ejercicio que crea una clase para la creación de hilos mediante Runnable que se comporta como implementada con Thread con un parámetro de número de hilo.

(Ver 2.3).

f. Amplia el ejercicio: **1-c** añadiendo que el constructor reciba un segundo parámetro **n** que será el número de veces que se presentará un mensaje implementado en el método **run**, dentro de un **for**. Modifica el nombre a: **MiHilo6**. Añade un atributo **prívate** de tipo **int** y nombre **n** que tomará valor en el constructor. Modifica el mensaje a mostrar para que aparezca el número de hilo y su número en el bucle. En cada pasada del bucle haz que espere durante 1 segundo. Ejercicio donde se incluye el método sleep para parar el hilo 1 segundo tras cada presentación en pantalla de un mensaje mediante Runnable.

g. Amplia el ejercicio: **1-d** añadiendo que el constructor reciba un segundo parámetro **n** que será el número de veces que se presentará un mensaje implementado en el método **run**, dentro de un **for**. Modifica el nombre a: **MiHilo7**. Añade un atributo **prívate** de tipo **int** y nombre **n** que tomará valor en el constructor. Modifica el mensaje a mostrar para que aparezca el número de hilo y su número en el bucle. En cada pasada del bucle haz que espere durante 1 segundo. Ejercicio donde se incluye el método sleep para parar el hilo 1 segundo, tras cada presentación en pantalla de un mensaje, mediante Thread.

(Ver 2.4).

h. Modifica el ejercicio: **1-f** cambiando sleep por yield (). Modifica el nombre a: **MiHilo8**. En cada pasada del bucle haz que se use yield para ceder el paso a otro hilo que esté a la espera de entrar en ejecución. Ejercicio donde se incluye el método yield para parar el hilo y ceder el paso a otro hilo preparado para ejecución mediante Runnable.

i. Modifica el ejercicio: **1-g** cambiando sleep por yield (). Modifica el nombre a: **MiHilo9**. En cada pasada del bucle haz que se use yield para ceder el paso a otro hilo que esté a la espera de entrar en ejecución. Ejercicio donde se incluye el método yield para parar el hilo y ceder el paso a otro hilo preparado para ejecución mediante Thread.

(Ver 2.5).

j. Modifica el **1-f** que aunque ya está preparado para ser interrumpido desde el exterior al contener en su catch el **return** que hace que el hilo esté preparado para ser interrumpido, le añadiremos un mensaje de adiós con el número de hilo e indicando que ha sido interrumpido. Modifica el nombre a: **MiHilo10**. Modifica el tiempo de sleep a 100L. Ejercicio para saber que podremos interrumpir un hilo siempre que esté preparado para ello mediante el return que hace finalizar el hilo.

k. Modifica el **1-h** ya que al parar el hilo con yield se debe preparar para ser interrumpido desde el exterior. Para ello hay que incluir el **return** dentro de una condición que compruebe en cada pasada si el hilo ha sido interrumpido usando: **isInterrupted ()**.Modifica el nombre a: **MiHilo11**. Ejercicio para preparar un hilo para que pueda ser interrumpido cuando usamos yield mediante el return que hace finalizar el hilo y la condición de comprobación isInterrupted ().

(Ver 2.6).

l. Crea una clase, **Sumador**, que cree un hilo mediante la implementación la interface: **Runnable**. Dos atributos **privados**: int **n** y ContadorCompartido **o**. En el constructor asigna n y o (n es el número de veces que se ejecutará el hilo y **o** es el objeto compartido). En el método run crea el for y dentro una llamada a: o.sumar() y una pausa mediante sleep de 10L. Prepara el hilo para poder ser interrumpido. Ejercicio que crea una clase para la creación de hilos mediante Runnable.

m. Crea una clase, **Restador**, que cree un hilo mediante la implementación la interface: **Runnable**. Dos atributos **privados**: int **n** y ContadorCompartido **o**. En el constructor asigna n y o (n es el número de veces que se ejecutará el hilo y **o** es el objeto compartido). En el método run crea el for y dentro una llamada a: o.restar() y una pausa mediante sleep de 10L. Prepara el hilo para poder ser interrumpido. Ejercicio que crea una clase para la creación de hilos mediante Runnable.

(Ver 3.1).

2. Clases de creación y ejecución de hilos.

2.1. Crea una clase principal, **Test1**, en cuyo **main** se creen dos hilos, uno para el **1-a**, otro para el **1-b** y la ejecución de los mismos. Tras la creación de ambos pon en marcha los dos hilos. Ejercicio para comprobar la creación de hilos mediante diferentes métodos y su puesta en ejecución.

2.2. Crea una clase principal, **Test2**, en cuyo **main** se creen dos hilos, uno para el **1-c**, otro para el **1-d**. Tras la creación de ambos pon en marcha los dos hilos. Ejercicio para la creación de hilos mediante diferentes métodos pasándole un número de hilo y su puesta en ejecución.

(Ver 1-e).

2.3. Crea una clase principal, **Test3**, en cuyo **main** se cree un hilo para el **1-e** y se ponga en marcha. Ejercicio para comprobar el uso de la creación de un hilo mediante Runnable que se comporta como creado con Thread.

(Ver 1-f).

2.4. Crea una clase principal, **Test4**, en cuyo **main** se creen dos hilos, uno para el **1-f**, otro para el **1-g**. Tras la creación de ambos pon en marcha los dos hilos. Ejercicio para comprobar el uso de sleep() implementado en el método run de las clases que crean el hilo.

2.5. Crea una clase principal, **Test5**, en cuyo **main** se creen dos hilos, uno para el **1-h**, otro para el **1-i**. Tras la creación de ambos pon en marcha los dos hilos. Ejercicio para comprobar el uso de yield() implementado en el método run de las clases que crean el hilo.

(Ver 1.j).

2.6. Crea una clase principal, **Test6**, en cuyo **main** se creen dos hilos, uno para el **1-j**, otro para el **1-k**. Tras la creación de ambos pon en marcha los dos hilos. Suspende el hilo principal, el propio main **1 milisegundo** y déjalo preparado para que pueda ser interrumpido desde el exterior, implementando en el **catch** el **return**. Tras ese milisegundo, interrumpe el hilo número 1. Ejercicio para comprobar el uso de interrupciones de hilos.

2.7 Crea una clase principal, **Test7**, en cuyo **main** se creen dos hilos, uno para el **1-j**, otro para el **1-k**. Tras la creación de ambos pon en marcha los dos hilos. Usa **join ()** para que el hilo 2 entre en ejecución desplazando al que se esté ejecutando. Tras ello añade **un mensaje de que el hilo 2 ha terminado**, debido a que join hará que hilo 2 se ejecute hasta su finalización o se interrumpa desde el exterior. Por último añade una línea para interrumpir el hilo 1 (comprueba el resultado con esta última línea y también con ella comentada).
Ejercicio

2.8. Crea una clase principal, **Test8**, en cuyo **main** se creen dos hilos, uno para el **1-l**, otro para el **1-m**, llamados sumador y restador, con parámetros: **10**, **o**. La clase tendrá un atributo **static ContadorCompartido o**. Al entrar en el main se asigna a **o** un **new ContadorCompartido ()**. Tras la creación de ambos hilos pon en marcha los dos hilos. Usa **join ()** con sumador y a continuación, join con restador. Finaliza con un mensaje que indique el valor de o.cont. Ejercicio para comprobar que variables que usadas por varios hilos y que no están sincronizadas proporcionan resultados erróneos. Se supone que si sumamos los primeros 10 valores y restamos los mismos valores el resultado debe ser cero.

(Ver 3.2).

3. Sincronización.
   3.1. Crea una clase, llamada: **ContadorCompartido** que tenga como atributo **público** un **int** llamado: **cont** con valor **0**. Añade dos métodos **públicos**: **sumar** y **restar** que no reciben parámetros y dentro de ellos la operación que suma 1 a cont y en el otro la operación que resta 1 a cont. Ambos terminan con un mensaje indicando si suman o restan y el valor de cont.

(Ver 2.8).

3.2. Modifica el 3.1 para que los métodos sumar y restar tengan en su signatura el modificador: **synchronized**. Ejecuta de nuevo el 2.8 y comprueba ahora el resultado.