UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

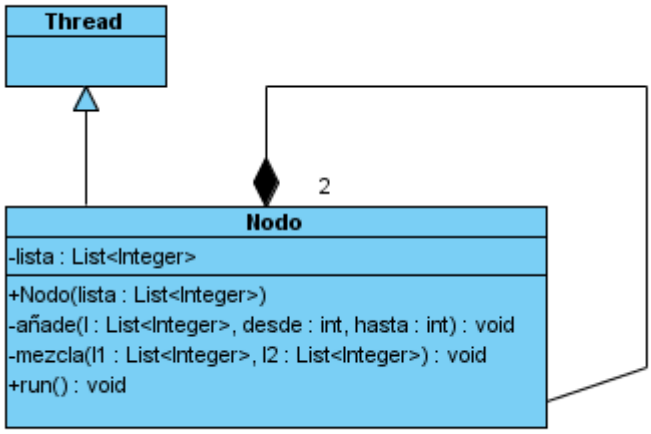
Programación de Sistemas y Concurrencia Práctica nº4.

1. Realizar un programa en Java con 3 hebras, cada una de las cuales escribe por pantalla varias veces (valor pasado como parámetro en el constructor) el carácter que se le indique (también indicado como parámetro). ¿Se mezclan las letras? Justifica el comportamiento observado.

2. Disponemos de una clase denominada `VariableCompartida` que encapsula el valor de una variable `v` de tipo `int`. La clase `VariableCompartida` contiene métodos para establecer (método `set`), obtener (método `get`) o incrementar (método `inc`) el valor de `v`. Realizar un programa en Java que cree 2 hebras compartiendo una instancia de la clase `VariableCompartida` e incrementen cada una de ella 10 veces el valor de `v`. Mostrar desde la hebra del programa principal el valor final de `v`. ¿Se obtienen los resultados esperados? Aumenta progresivamente el número de incrementos hasta observar algún comportamiento "extraño". Justifica los resultados obtenidos.

3. Utilizando la clase `VariableCompartida` del ejercicio anterior y una instancia compartida de dicha clase, realizar un programa en Java donde disponemos de una hebra que modifica los valores de la instancia de `VariableCompartida` desde 0 a 99 utilizando el método `set`. Disponemos también de otra hebra, que utilizando el método `get` debe mostrar por pantalla todos los cambios que se van produciendo en la instancia de `VariableCompartida` (sabiendo que se van a generar 100 valores). ¿Se recogen todos los valores? ¿Qué ocurre? Intenta solucionar los problemas detectados.

4. Diseña un programa java que implemente el algoritmo recursivo de ordenación por mezcla de forma concurrente. El programa debe construir un árbol binario de hebras, de profundidad variable, dependiente del número de elementos de la lista a ordenar. El sistema sólo necesita una clase **Nodo** como la descrita en el diagrama siguiente:



Inicialmente, el método **main()** crea un primer nodo (el nodo raíz del árbol) al que se le pasa una lista de números aleatoria. Cada uno de los nodos que se vayan creando en el sistema se comporta de la misma forma. Si la lista que se le pasa al nodo en el constructor tiene 0 o 1 elementos, no hay nada que hacer (la lista ya está ordenada). En otro caso, la hebra crea dinámicamente dos nuevas hebras, y le pasa a cada una de ellas una de las dos mitades de su lista. Una vez que cada hebra hija ha ordenado sus mitades, la hebra padre las mezcla de forma ordenada para producir el resultado esperado. Como ayuda para la implementación del método **run(),** se sugiere implementar los métodos privados **añade(l,d,h)** que añade a la lista **l** todos los elementos de **lista** desde la posición **d** (incluida) hasta la posición **h** (excluida), y **mezcla(l1,l2)** que mezcla de forma ordenada los elementos de las listas **l1** y **l2** dejando el resultado en **lista**.

Nota: utiliza la interfaz *List<Integer>*, y la clase *ArrayList<Integer>* para representar las listas manejadas por tu programa. Para facilitar la implementación puedes usar los métodos *add*, *addAll*, *subList* de estas clases.

5. Implementar un programa en Java que permita calcular términos de la sucesión de Fibonacci. Se dispondrá de N hebras para calcular hasta el término N-ésimo de dicha sucesión, de forma que la hebra con identificador i-ésimo debe calcular el término i-ésimo utilizando los valores de las hebras i-1 e i-2. Las hebras deben esperar a que estén calculados los términos anteriores al suyo. Considerar los términos 0 y 1 como casos especiales.