Práctica 4
Concurrencia en Java

Escuela de Ingeniería y Arquitectura
Depto. de Informática e Ingeniería de Sistemas

1. Objetivos

1. Crear y manipular hilos de ejecución en Java
2. Utilizar monitores en Java

2. Contenidos

Esta práctica consta de un total de tres partes. La primera de ellas trata sobre hilos de ejecución en Java y su manipulación (a través de la clase `Thread` y la interfaz `Runnable`). La segunda parte está dedicada a la exclusión mutua (métodos **`synchronized`**) y monitores (métodos `wait()`, `notify()` y `notifyAll()` de la clase `Object` de Java). Por último, en la tercera se plantea de nuevo el problema del club de golf presentado en la práctica anterior, pero solicitándose su implementación en Java.

Los ficheros con código fuente de Java a los que se hace referencia en esta práctica están disponibles en la página de la asignatura: `http://webdiis.unizar.es/~ezpeleta/doku.php?id=practicas_0708_pc`. La práctica puede realizarse en cualquier entorno donde esté instalado Java, como *hendrix-ssh* o los entornos *Windows* de los laboratorios del Departamento de Informática e Ingeniería de Sistemas, donde hay además instalados entornos de desarrollo para Java, como *jGRASP*[1] y *Eclipse*[2].

[1] `http://www.jgrasp.org/`
[2] `http://www.eclipse.org/`

3. Descripción del trabajo a realizar

3.1. Ejecución concurrente en Java

El directorio «`parte1`» contiene dos ficheros con las definiciones de dos clases: `Main` y `TareaSimple`. El constructor de la clase `Main` crea 10 hilos de ejecución de tipo `TareaSimple` y los activa. La clase `TareaSimple` implementa la interfaz `Runnable` y, cuando es activada, escribe en pantalla un total de `TareaSimple.NUM_VECES` veces un dígito `d` comprendido entre el 0 y el 9 que se le ha indicado en el constructor e intenta (aunque no lo consiguen) que las líneas escritas en la pantalla sean de exactamente `Main.LONG_LINEA` caracteres. Para ello, los objetos de la clase `TareaSimple` actualizan y consultan el valor del atributo `numEscritos` de la clase `Main` a través de los métodos `incrementarNumEscritos()` y `numEscritos()`. El diagrama UML de estas clases puede verse en la figura 1.

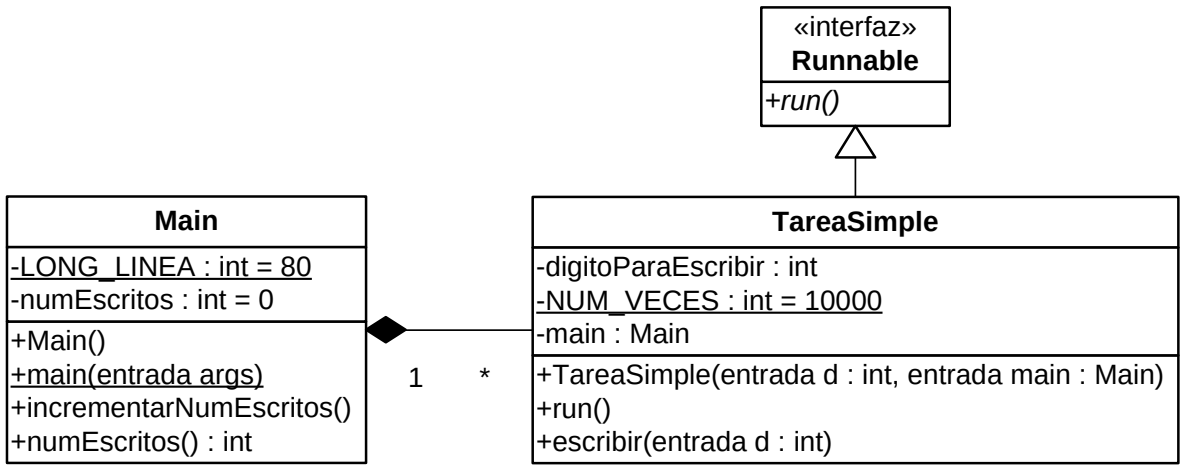

Figura 1: Diagrama de clases de la primera parte de la práctica

Estudia el código de los ficheros «`Main.java`» y «`TareaSimple.java`» y arregla el problema de que no todas las líneas escritas sean de exactamente `Main.LONG_LINEA` caracteres. Como paso previo, modifica el código del constructor de la clase `Main`, de forma que informe del valor que el atributo `numEscritos` tiene cuando finaliza la ejecución de todos los hilos.

Documenta en un fichero denominado «`cuarta.txt`» los cambios realizados en el código para corregirlo, tanto si han contribuido a solucionar el problema como si no, e interpreta el comportamiento del programa tras cada cambio.

3.2. Sincronización y monitores en Java

Crea un nuevo directorio denominado «`parte2`» y copia en él los ficheros «`Main.java`» y «`TareaSimple.java`» modificados tras la realización de la parte anterior, con el objeto de seguir cambiándolos. Se trata ahora de sincronizar los hilos de ejecución para que las escrituras de los dígitos se vayan haciendo por orden (del 0 al 9 y así sucesivamente). Para ello, convierte la clase `Main` en un monitor que se encargue de controlar qué tarea tiene el turno para escribir. Comprueba los diferentes comportamientos del programa en los casos de utilizar `notify()` o `notifyAll()` y documéntalos en el fichero «`cuarta.txt`».

Aún sin necesidad de medir de forma explícita los tiempos de ejecución del programa resultado de esta parte y del de la anterior, ¿existe alguna diferencia de tiempo ejecución total evidente entre ambas? Si se desea responder con más precisión a esta pregunta, el método estático `System.currentTimeMillis()` devuelve la hora actual del sistema expresada como una cantidad de milisegundos a partir de un momento fijo en el tiempo y puede servir de base para calcular los tiempos de ejecución de la parte primera y segunda.

3.3. El problema del club de golf

En la práctica anterior, se presentó el problema de un club de golf que únicamente permite jugar con los palos y pelotas que él mismo alquila. La cantidad de material disponible en el club es reducida: concretamente, 20 pelotas y 20 palos de golf. Como paso previo a jugar, cada socio del club debe solicitar el alquiler del material necesario. Hay dos clases de socios: novatos y con experiencia. Los jugadores con experiencia siempre alquilan una única pelota y entre 2 y 5 palos para jugar. Un jugador novato suele alquilar un número mayor de pelotas (nunca más de 5 pelotas) y dos únicos palos.

Se trata ahora de desarrollar un programa Java que simule la actividad que un grupo de jugadores lleva a cabo en las instalaciones del mencionado club. Igual que en la práctica anterior, habrá 7 jugadores con experiencia y otros 7 novatos. Cada jugador se identificará a través de un número entero comprendido entre 1 y 14, seguido de un signo '+' si se trata de un jugador con experiencia y de un signo '−' si se trata de un jugador novato.

Se recuerda que cada jugador muestra, repetidamente, el siguiente comportamiento, independientemente de su nivel de experiencia: decide jugar y reserva el material necesario, juega durante un tiempo, devuelve el material que ha usado, y, finalmente, descansa durante un rato. En ningún caso podrá comenzar a jugar si no dispone del material necesario. Del mismo modo, no iniciará su descanso hasta haber devuelto el material usado para jugar. Cada jugador deberá escribir en la pantalla mensajes indicando su comportamiento:

- Antes de realizar una reserva deberá escribir un mensaje donde muestre su identi-

dad y el número de recursos solicitados.

- Una vez haya conseguido reservar el material que necesita, escribirá un mensaje indicando su identidad, los recursos alquilados y su intención de comenzar a jugar.
- Antes de realizar una devolución deberá escribir un mensaje donde muestre su identidad y los recursos devueltos.
- Una vez completada la devolución, escribirá un mensaje indicando su identidad, los recursos devueltos y su intención de descansar.

Para dar homogeneidad al formato de los mensajes, la representación de la identidad de un jugador y el material que reserva/usa/va a devolver/ha devuelto se ajustará al siguiente formato: "*identidad[número de pelotas, números de palos] acción que realiza*". Por ejemplo, "`2+[1,3] reserva`" representa que el jugador con identificador 2 (con experiencia) reserva 1 pelota y 3 palos.

3.3.1. Pautas de implementación

Para implementar el simulador anterior, se facilita el esqueleto de tres clases: `Simulador`, `Jugador` y `Club`.

La clase `Club` implementa un monitor que gestiona la concurrencia en el uso del material disponible en el club. Como mínimo, deberá encapsular dos atributos privados que representen el número de pelotas y palos de golf disponibles y tendrá que disponer de los métodos necesarios para realizar su reserva y devolución. Cada objeto de la clase `Club` se podrá configurar fácilmente en cuanto a la cantidad de recursos inicialmente disponibles.

La clase `Jugador` implementa los procesos del sistema. No es necesario definir una clase diferente para cada tipo de jugador. El constructor de objetos de la clase `Jugador` permite que el programa principal indique cuál es el identificador del jugador que se está creando y el tipo de jugador al que pertenece. El comportamiento que exhibe un jugador (reservar material, jugar, devolver material y descansar) deberá ser repetido por cada tarea un número de veces concreto que también será indicado por el programa principal a través del constructor. El tiempo que dedica a jugar y descansar será aleatorio (no superior a un segundo para cada actividad) y podrá ser diferente en cada ocasión.

Finalmente, la clase `Simulador` tiene el método `main()` que, al ser ejecutado, constituye el programa principal Java, encargado de configurar y ejecutar el sistema. Creará los elementos que lo constituyen: jugadores, club y material disponible inicialmente. En una primera versión, configurad los 14 jugadores (7 con experiencia y 7 novatos) para que repitan su comportamiento 5 veces. Se recomienda experimentar cómo se comportaría el sistema con otras configuraciones alternativas, por ejemplo, modificando el número de cada tipo de jugadores, la cantidad inicial de material, el número de iteraciones de comportamiento de cada jugador, etc.

Como base para obtener tiempos de espera aleatorios, se puede utilizar el método `Math.random()`.

El esqueleto de las clases cuya implementación se propone se encuentran en la carpeta «`parte3`».

4. Entrega de material

- Como resultado de esta práctica hay que enviar por correo electrónico al profesor de prácticas (`latre@unizar.es`) un fichero comprimido en formato ZIP denominado «`pract4.zip`» que contenga un directorio (paquete) por cada una de las partes de la práctica con las clases Java solicitadas en cada una de ellas y el informe denominado «`cuarta.txt`».
- Fecha límite de entrega: 23 de mayo de 2011.