Objetivos Hilos y sincronización

Índice

1. Hilos

Vamos a ver varias posibilidades a la hora de crear hilos:

1.1. Extendiendo `Thread`

La clase `Thread` contiene un metodo `run()` genérico que no realiza ninguna tarea. Extendiendo la clase `Thread` con nuestra clase deberemos sobreescribir el método `run()` de tal forma que realice la tarea que deseamos: ordenar datos, realizar alguna animación, ... El método `run()` es llamado cuando llamamos al método `start()` heredado de `Thread`.

Ejemplo: Un hilo que calcula en resto de la división, según el estándar IEEE 754, entre un número dado y todos los números comprendidos entre 1 y el número dado :

```
class Resto extends Thread{
   private int maximo;

   Resto(int max){
      maximo=max;
   }

   public void run(){
      for (int i=1 ; i <= maximo; i++)
         System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
   }
}
```

En otra parte del programa creamos los hilos y los ejecutamos:

```
Resto r1 = new Resto(100);
Resto r2 = new Resto(200);
r1.start();
r2.start();
...
```

1.2. Implementando `Runnable`

La interfaz `Runnable` contiene un único método: `run( )` Una clase que implemente la interfaz Runnable tiene que implementar el método `run( )`. Uno de los constructores de la clase Thread

permite la creación de un hilo a partir de un objeto que implemente la interfaz Runnable del siguiente modo:

```
Thread h = new Thread(Runnable target)
```

Cuando se crea un hilo de este modo, el Thread obtiene el método `run( )` del objeto que implementa a la interface `Runnable`. El ejemplo anterior sería ahora:

```
class Resto implements Runnable{
   private int maximo;

   Resto(int max){
      maximo=max;
   }

   public void run(){
      for (int i = 1; i <= maximo; i++)
         System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
   }
}
```

Y donde queramos crear y ejecutar los hilos:

```
...
Thread r1 = new Thread(new Resto(20));
Thread r2 = new Thread(new Resto(100));
r1.start();
r2.start();
...
```

Otra posibilidad es lanzar el hilo desde el constructor de `Resto`:

```
class Resto implements Runnable{
   private int maximo;

   Resto(int max){
      maximo=max;
      new Thread(this).start();
   }

   public void run(){
      for (int i = 1; i <= maximo; i++)
         System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
   }
}
```

En otra parte del programa creando los objetos del tipo `Resto` se lanzan los hilos.

```
...
new Resto(100);
new Resto(200);
...
```

Finalmente, si queremos tener una referencia al hilo dentro de la clase `Resto` (para usarla en otra parte) modificaríamos la clase del siguiente modo:

```
class Resto implements Runnable{
   private int maximo;
   private Thread hilo;

   Resto(int max){
      maximo=max;
      hilo = new Thread(this);
      hilo.start();
   }

   public void run(){
      for (int i = 1; i <= primmax; i++)
```

```
            System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
        }
        ...
    }
```

2. Sincronización y monitores

La sincronización es necesaria para evitar colisiones entre hilos cuando todos ellos comparten el acceso a un determinado recurso. Hay dos modos de sincronización:

- En una clase se puede sincronizar un método para evitar que cuando un hilo haya llamado al método ningún otro hilo pueda llamar a ningún otro método sincronizado de la clase.

  `synchronized metodo(...)`

- Se puede sincronizar el acceso a un objeto a nivel de bloque de código:

  `synchronized(objeto)...`

Un monitor es un objeto que implementa el acceso en exclusión mutua a sus métodos. En Java se aplica a los métodos de la clase declarados como `synchronized`. Los métodos que no sean declarados como `synchronized` pueden ser utilizados concurrentemente.

Dentro de un método `synchronized` se pueden utilizar los métodos (heredados de `Object`): `wait()`, `notify()` o `notifyAll()`. Una descripción de estos métodos se muestra en el cuadro 1.

3. Tareas

Se trata de realizar un análisis del contenido de una serie de páginas web mediante un número de hilos productores y un hilo consumidor que integra los resultados parciales que van proporcionando los productores. Un esquema general del funcionamiento del programa se muestra en la figura 1.

El listado de páginas web a analizar se encuentra en un fichero, donde cada línea contiene una dirección URL. Las páginas se analizarán con un número fijo de hilos (productores) el número de hilos a utilizar se pasará como un argumento al programa.

Para cada página el análisis consiste en encontrar las palabras diferentes y contarlas. La estructura utilizada para almacenar esta información es un `HashMap` donde la clave es la palabra y el valor es un objeto que contiene un atributo de tipo `long` que tiene el número de ocurrencias de la palabra.

Al finalizar el análisis de la página cada hilo almacena el `HashMap` que ha obtenido en un *buffer*.

Habrá un único hilo consumidor cuya misión es mantener un `HashMap` actualizado utilizando la información que van insertando los productores. Cada vez que el consumidor actualiza su `HashMap` muestra la siguiente información:

1. Cual es la palabra más repetida hasta el momento.
2. El número de apariciones hasta el momento.
3. El tanto por ciento respecto al número total de palabras procesadas hasta el momento.
4. El número de direcciones procesadas hasta el momento.
5. El número total de palabras procesadas hasta el momento.

Método	Descripción
`wait()`	una llamada a este método causa que el hilo (llamémosle H) que posee el monitor espere hasta que otro hilo llame a notify() o notifyAll() (si no se le pasan argumentos), o hasta que transcurra un determinado tiempo (si se le pasa un argumento con el tiempo). El hilo queda deshabilitado para propósitos de planificación de hilos y está dormido hasta que ocurra alguna de las siguientes situaciones: ▪ algún otro hilo invoca al método notify() de este objeto y el hilo H es elegido para despertar. ▪ Otro hilo llama al método nofifyAll() del objeto. ▪ Otro hilo interrumpe al hilo H. ▪ Si ha transcurrido el tiempo pasado en el argumento. El hilo H se quita del conjunto de hilos en espera y se le habilita para la planificación de los hilos compitiendo con los demás para sincronizar el objeto. Una vez que gana el control sobre el objeto continua al estado en el que estaba antes de llamar al wait(). Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto. Un hilo se convierte en el propietario del monitor del objeto de alguna de las siguiente formas: ▪ Ejecutando un método synchronized del objeto. ▪ Ejecutando el cuerpo de una sentencia synchronized que sincroniza el objeto. ▪ Para objetos del tipo Class ejecutando un método estático synchronized de esta clase.
`notify()`	Despierta un único hilo que esté esperando en el monitor de este objeto. Un hilo espera para tomar el monitor de un objeto llamando a uno de los métodos wait(· · ·). Si hay varios hilos esperando, se elige a uno de ellos. La elección es arbitraria (no se toma por ejemplo el que lleva más tiempo esperando). El hilo despertado debe competir con otros hilos por obtener el monitor ya que no tiene privilegios especiales para bloquear el objeto. Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto (ver wait()).
`notifyAll()`	Despierta a todos los hilos que están esperando para tomar el monitor de este objeto. Un hilo espera para tomar el monitor de un objeto llamando a uno de los métodos wait(· · ·). Los hilos despertados competirán para sincronizar este objeto. Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto (ver wait())

Cuadro 1: Métodos para que un objeto controle la sincronización de los hilos. Se heredan de `Object`

Un ejemplo de salida se muestra en la figura 1.

Hemos de tener un buffer donde se almacenan los resultados que generan los hilos productores. Esta clase tiene un método para almacenar y otro para recuperar. La recuperación se realiza de forma secuencial, el hilo consumidor no puede avanzar en una posición si en esa posición un productor no ha escrito su resultado. Cada vez que un hilo productor finalice lo debe notificar al buffer.

En el momento de crear un hilo productor hay que pasarle el buffer donde debe guardar la información y un una referencia del tipo `AccesoFichero` (que es un acceso a fichero sincronizado) de donde debe obtener las direcciones URL.

3.1. Clases proporcionadas

Se proporcionan las siguientes clases:

- `Productor` clase que extiende a `Thread` y que en su método `run()` analiza el texto contenido en una página web y almacena en el buffer los `HashMap` que va construyendo.
- `Contador` clase que representa un contador de apariciones.
- `AccesoFichero` clase que representa el acceso a un fichero de forma sincronizada.

Estas clases están en el fichero `clasesP5.jar` y su documentación en el fichero `clasesP5.zip`. Además se proporciona el fichero `Analisis.java` con la clase que contiene el main. El fichero `urls.txt` contiene el listado de direcciones a analizar (tomadas de `http://www.gutenberg.net`).

3.2. Clases a realizar

Se deben realizar las siguientes clases:

- `HiloConsumidor` clase que extraiga información del buffer, mantenga un `HashMap` actualizado y muestre la información parcial (tal y como se ha descrito y se muestra en la figura 1).
- `Almacen` que represente el buffer donde los productores almacenen la información y el consumidor la extraiga. Debe tener los siguientes métodos:
  - `almacena(HashMap hm)` que utilizan los productores
  - `HashMap extrae()` que utiliza el consumidor
  - `notificarTerminado()` que utilizan los productores para indicar que han finalizado.
  - `boolean finTodos()` que utiliza el consumidor para conocer si todos los hilos han finalizado o no.

  Hay que decidir cuales de estos métodos son `synchronized`.

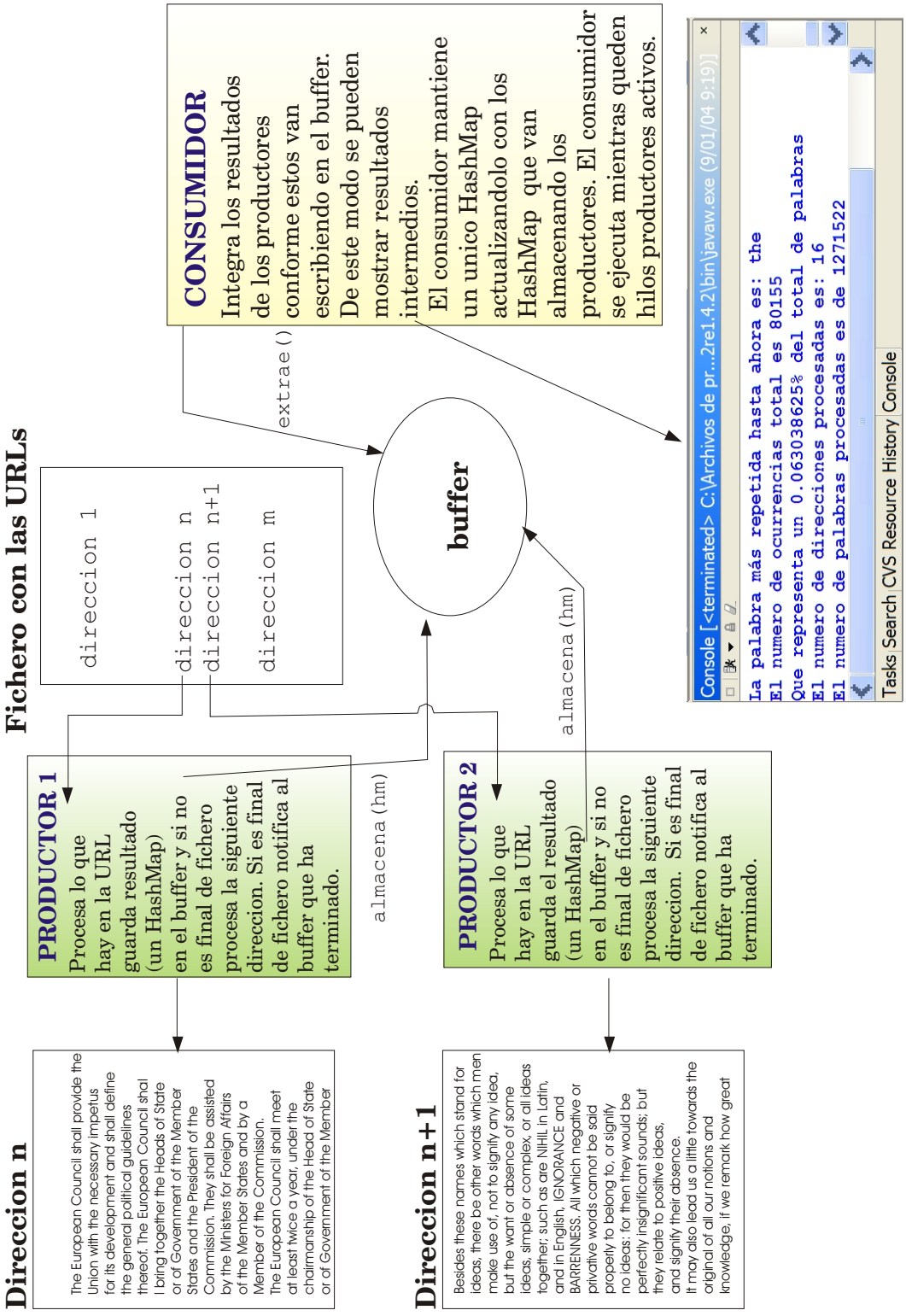

Figura 1: Esquema del funcionamiento del programa