Prácticas de laboratorio
Problema de las hormigas
(1 sesión)

Concurrencia y Sistemas Distribuidos

Introducción

El objetivo de esta práctica es analizar y completar un programa concurrente en el que se aplican distintas condiciones de sincronización entre hilos, haciendo uso de algunas de las herramientas Java proporcionadas en la biblioteca java.util.concurrent.

- Uso de ReentrantLock.
- Generación y uso de las Condiciones asociadas a un determinado ReentrantLock
- Aplicar soluciones para el problema de los interbloqueos

Esta práctica se desarrolla en una única sesión. Tenga en cuenta que necesitará destinar algo de tiempo de su trabajo personal para concluir la práctica. A lo largo de la práctica debe completar una serie de ejercicios. Se recomienda resolverlos y anotar sus resultados para facilitar el estudio posterior.

El problema de las hormigas

Se dispone de un territorio en el que vive un conjunto de hormigas. El territorio se modela como una matriz rectangular NxN, siendo N un parámetro del programa. Por su parte, cada hormiga se implementa como un hilo Java, con una posición inicial aleatoria dentro del territorio.

Cada hormiga se mueve libremente por el territorio: cada movimiento se realiza a una celda contigua a la que se encuentra (arriba, abajo, izquierda o derecha). Se puede indicar la cantidad inicial de hormigas, así como el número de movimientos que realiza cada una antes de terminar. Cuando una hormiga termina, desaparece del territorio (liberando la celda que ocupaba).

Las hormigas se modelan mediante la clase Ant, que extiende a Thread. El código básico de las hormigas es:

```
class Ant extends Thread {
   ...
   public void run() {
      ...
      t.hi(a); // t = terrain. put ant a in a random cell
      while (movs > 0) {
         movs--;    // decrement remaining moves
         t.move(a); // move ant a to random next cell
         delay();   // applies a random delay
      }
      t.bye(a);     // ant a disappears
      ...
   }
}
```

Como restricción a los movimientos de las hormigas, en cada celda del territorio puede haber como máximo una única hormiga:

- Inicialmente el programa sitúa a cada hormiga en una celda libre
- Si una hormiga desea moverse a una celda ocupada, debe esperar a que quede libre

Resolvemos el problema utilizando el concepto de monitor. Cuando una hormiga quiere desplazarse a una celda y ésta se encuentra ocupada, deberá suspenderse en una variable *condition* hasta que sea reactivada por otra hormiga. El interface Terrain modela el interfaz de dicho monitor:

```
interface Terrain {
   void hi  (int a);
   void bye (int a);
   void move(int a) throws InterruptedException;
}
```

Los métodos *hi, bye, move* se invocan desde el código de cada hormiga. La simulación termina cuando se cumple alguna de las siguientes condiciones:

1. Todas las hormigas han terminado su ejecución y abandonado el territorio.
2. Se ha llegado a una situación de interbloqueo, de forma que las hormigas que quedan vivas nunca podrán terminar.

Para modelar el territorio planteamos distintas clases que implementan Terrain:

- **Terrain0**.- Monitor básico (lock y condition implícitos).
- **Terrain1**.- Monitor general (java.util.concurrent) con una única variable condition para todo el territorio.
- **Terrain2**.- Monitor general con una variable condition por celda del territorio: una hormiga se suspende en la variable condición asociada a la celda ocupada a la que quiere desplazarse.
- **Terrain3**.- Monitor general con una variable condition por celda del territorio. Incluye un mecanismo para resolver el problema de los interbloqueos.

La clase Terrain0 ya está implementada, y debe utilizarse como punto de partida para los restantes tipos de territorio. El siguiente apartado presenta el código proporcionado: el resto de los apartados del boletín corresponden a las distintas actividades que se plantean, una por tipo de territorio.

Código proporcionado

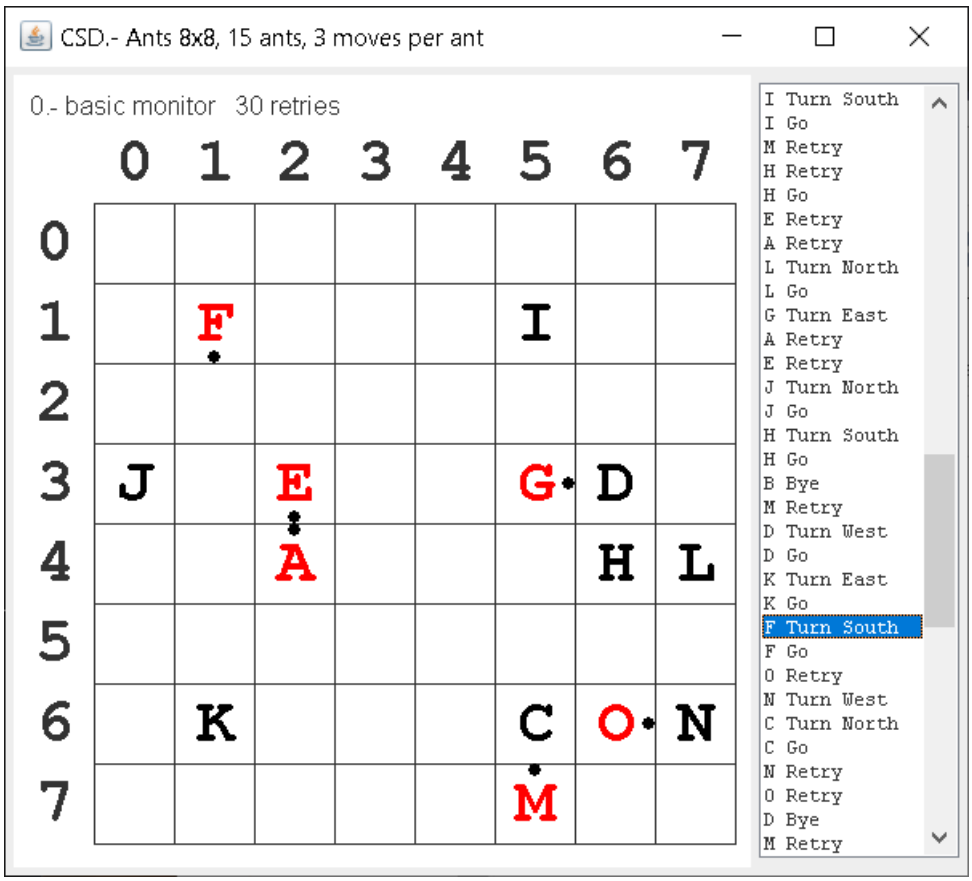
Puede descargar el código necesario para la práctica desde el Poliformat de la asignatura (fichero Ants.java).
La clase Ants contiene el método principal main. Desde un terminal, escribe java Ants (y posiblemente argumentos opcionales). Los argumentos posibles son:

- Tipo de Territorio (0,1,2,3, por defecto 0).
- Talla del territorio (entero entre 6 y 10, por defecto 8).
- Número de hormigas (entero entre 10 y 26, por defecto 15).
- Número de movimientos por cada hormiga (entero entre 2 y 5, por defecto 3).

Ejemplos:

- `java Ants` (Terrain0, 8x8 celdas, 15 hormigas, 3 movimientos).
- `java Ants 1 10 26 2` (Terrain1, 10x10, 26 hormigas, 2 movimientos).

Al ejecutar la aplicación, se muestra en pantalla una ventana con el territorio y una lista de eventos. Al seleccionar cualquier evento de la lista se muestra el correspondiente estado en el Territorio. La siguiente figura muestra el aspecto de la aplicación tras seleccionar un evento (`F Turn South`):



Las claves para interpretar la figura son:

- La cuadrícula de la izquierda representa el estado del territorio (inicialmente vacío), y la lista de la derecha la secuencia de eventos. El contenido del territorio representa la situación en el punto seleccionado en la lista de eventos (inicialmente ninguno).

- Al seleccionar un evento de la lista se muestra el estado del territorio. La selección de eventos es libre (adelante, atrás, salto al evento deseado, ...).
- Una celda del territorio puede estar en blanco (vacía) o contener una hormiga, representada con una letra.
- Si la hormiga quiere desplazarse a otra celda aparece en color rojo y con un punto en la dirección en la que desea desplazarse. Toda hormiga que no se está desplazando se dibuja en negro.
- Para la lista de eventos se utiliza la siguiente notación:

Evento	**Significado**
`X hi (f,c)`	La hormiga X llega al territorio, y se sitúa en la posición (fila,columna).
`X turn dir`	La hormiga X quiere desplazarse a la celda vecina en la dirección dir (North, West, South, East).
`X go`	La hormiga X completa el movimiento (se desplaza a la casilla vecina correspondiente).
`X retry`	La hormiga X estaba esperando (destino ocupado) y reintenta el movimiento.
`X chgDir`	La hormiga X estaba esperando (destino ocupado), y ha decidido cambiar de dirección.
`X bye`	La hormiga X abandona el territorio (la casilla queda vacía).

Sobre la figura que muestra el resultado de la ejecución podemos interpretar lo siguiente:

- Hormigas que no intentan ningún movimiento: I, J, D, H, L, K, C, N.
- Hormigas que intentan un movimiento hacia un destino libre: F
  - Cuando el destino está libre, el siguiente evento corresponde al desplazamiento propiamente dicho (en la figura evento F go).
  - Únicamente puede haber como máximo una hormiga en esa situación
- Hormigas que intentan un movimiento hacia un destino ocupado: E, G, A, O, M.
  - E y A están en situación de interbloqueo: nunca podrán completar sus respectivos movimientos.
  - El resto de hormigas en espera (G,O,M) podrán completar su movimiento cuando queden libres las respectivas casillas destino.
- En el título de la ventana se indican los parámetros de la simulación
- En la parte superior aparece como mensaje el tipo de monitor y la cantidad de reintentos (eventos *retry*) realizados por parte de todas las hormigas durante la simulación completa.

Análisis del código

El código proporcionado implementa distintas clases. Ninguna de ellas debe modificarse

- *Pos, Op, Hi, Bye, Turn, Retry, Go, ChgDir, Viewer son* "clases opacas", necesarias para el correcto funcionamiento de la aplicación, pero sin interés para el alumno.
- *Ant es la* clase que extiende a Thread e implementa el concepto de hormiga. Debe comprenderse el funcionamiento de su método run (ya presentado).
- Interface *Terrain* (ya comentado).
- *Terrain0* es el punto de partida para diseñar el resto de territorios. Debe analizarse hasta comprender su funcionamiento. Las operaciones sobre v (atributo de tipo Viewer) son necesarias para el funcionamiento del programa: al desarrollar *Terrain1*, *Terrain2*, *Terrain3* deben mantenerse.

```
class Terrain0 implements Terrain {
   Viewer v;
   public Terrain0 (int t, int ants, int movs) {
      v=new Viewer(t,ants,movs,"0.- basic monitor");
      for (int i=0; i<ants; i++)
         new Ant(i,this,movs).start(); // lanza las hormigas
   }
   public synchronized void hi (int a) {v.hi(a); }
   public synchronized void bye (int a) {v.bye(a); }
   public synchronized void move (int a)
          throws InterruptedException {
      v.turn(a); Pos dest=v.dest(a);
      while (v.occupied(dest)) {
         wait();
         v.retry(a);
      }
      v.go(a);
      notifyAll();
   }
}
```

- No se proporciona el código de las clases Terrain1, Terrain2, Terrain3 (deben desarrollarse en esta práctica).
- *Ants* es la **clase principal**. Recoge los argumentos desde línea de órdenes y lanza las simulaciones utilizando los distintos tipos de territorio.

Actividad 0 (Sincronización básica en Java)

Lance varias ejecuciones del código proporcionado usando *Terrain0* y los valores por defecto (*java Ants*), y complete la siguiente tabla, donde se indique para cada ejecución:

- el número total de reintentos (retries) realizadas por las hormigas
- y la cantidad de hormigas que se han quedado al final bloqueadas.

Prueba	Total Retries	Nº hormigas interbloqueadas
1	11	0
2	64	5
3	58	4
4	60	3
5	43	2
6	8	0

Repita las ejecuciones con la orden *java Ants 0 8 10 3*

Prueba	Total Retries	Nº hormigas interbloqueadas
1		
2		
3		
4		
5		
6		

Indica, de media, cuántas esperas realizan en total las hormigas, cuándo:
a) La ejecución acaba de forma adecuada, sin interbloqueos.
b) Se produce un interbloqueo.

a) Media de esperas: 9,5
b) Media de esperas: 56,25

Actividad 1 (Sincronización con ReentrantLocks en Java)

Utilizando las herramientas *ReentrantLock* y *Condition* proporcionadas en *java.util.concurrent*, desarrolle *Terrain1*.

Rellene las tablas siguientes

java Ants 1

Prueba	Total Retries	Nº hormigas interbloqueadas
1	6	2
2	16	0
3	52	3
4	117	6
5	37	2
6	47	3

java Ants 1 8 10 3

Prueba	Total Retries	Nº hormigas interbloqueadas
1		
2		
3		
4		
5		
6		

¿Se aprecian diferencias significativas entre *Terrain0* y *Terrain1*?

- ¿se producen menos reactivaciones "innecesarias" de hilos en *Terrain1* que en *Terrain0*?

En cuanto a ejecuciones acabadas el número de esperas es mayor en el *reentrantLock* pero en las que se produce interbloquep no se aprecia la diferencia.

Indica, de media, cuántas esperas realizan en total las hormigas, cuándo:

a) La ejecución acaba de forma adecuada, sin interbloqueos.

b) Se produce un interbloqueo.

a) Media de esperas: 16

b) Media de esperas: 51,8

Actividad 2 (Uso de varias variables Condition)

Desarrolle *Terrain2*, utilizando un array de condiciones (tantas condiciones como celdas)
NOTA.- La sentencia *v.getPos(a)* devuelve la posición actual de la hormiga a.
NOTA.- A partir de una posición (ej. *Pos dest=v.dest(a);*) podemos acceder a sus componentes *x* e *y* (*dest.x* y *dest.y*)

Indica, de media, cuántas esperas realizan en total las hormigas, cuándo:
a) La ejecución acaba de forma adecuada, sin interbloqueos.
b) Se produce un interbloqueo.

Rellene las tablas siguientes

a) Media de esperas: 3
b) Media de esperas: 7,2

java Ants 2

Prueba	Total Retries	Nº hormigas interbloqueadas
1	8	5
2	5	2
3	7	3
4	8	7
5	3	0
6	8	3

java Ants 2 8 10 3

Prueba	Total Retries	Nº hormigas interbloqueadas
1		
2		
3		
4		
5		
6		

En esta implementación solo despertamos a una hormiga que sabemos que va a poder continuar

¿Se aprecian diferencias significativas entre Terrain0, Terrain1 y Terrain2?

Sí, hay una diferencia significativa debido a que ahora solo despertamos a la hormiga que sabemos que va a poder continuar.

(waiting for representa una hormiga que está esperando)

Actividad 3 (Gestión de interbloqueos)

Desarrolle *Terrain3* . *Terrain3* utiliza una condición por celda (como *Terrain2*), pero además resuelve el problema del interbloqueo. Cuando una hormiga completa un tiempo máximo de espera sin poder completar el movimiento (ej 300ms), realiza un cambio de dirección *v.chgDir(a);* (lo cual obliga a recalcular el destino);
NOTA.- *await(long timeout, TimeUnit unit)* bloquea hasta el signal correspondiente (en cuyo caso devuelve true) o bien hasta el vencimiento del periodo timeout indicado (en cuyo caso devuelve false)

Para: USAR BARRERAS

Indica cómo clasificar el mecanismo utilizado para resolver el problema de los interbloqueos (prevención, evitación, detección+recuperación), y en su caso la condición de Coffman que se rompe.

En todas las actividades anteriores se podría presentar en algún momento un problema de interbloqueo, si dos o más hormigas llegan a una espera circular.

Para resolver el problema del interbloqueo, se debe romper alguna de las condiciones de Coffman, excepto la de exclusión mutua (es decir, las hormigas deben de seguir respetando la restricción de no estar más de una en la misma celda del territorio).

Se puede resolver de varias formas diferentes:
-Se podría usar los métodos de tiò wait(long timeout, int nanos) o bien await(long timeout, TimeUnit unit) para esperar un máximo de tiempo.
-Se podría permitir que las hormigas pudieron ser "levantadas" del territorio y recolocadas en él posteriormente, para simular por ejemplo la expropiación de un recurso.
-Se podría establecer un lock por cada celda y hacer uso de los métodos tryLock(), tryLock(long timeout, TimeUnit unit) isLocked()...para determinar si una hormiga tiene que reconsiderar la celda a la cual desea ir. Se podría establecer un orden total de acceso en las celdas