Grado en Ingeniería Informática

Curso 2022/2023 – Convocatoria Ordinaria

03233102S – Gutiérrez Sánchez, Carlos

09087218X – Nieto Arias, Pablo

ÍNDICE

1) un análisis de alto nivel (descripción general del problema e identificación de sus principales actores);

2) diseño general del sistema y discusión de las herramientas de sincronización utilizadas;

3) descripción de las clases principales (atributos y métodos);

4) diagrama de clases; y

5) el código fuente, como anexo

**Análisis del problema**

En el enunciado se describe el funcionamiento de una colonia de hormigas. El programa deberá generar hormigas que entran a una colonia. Hay tres tipos de hormigas: las obreras, las soldados y las crías. Estos son elementos activos del problema. Cada uno de los tipos de hormigas tiene un comportamiento diferente. Además, en la colonia hay cinco zonas diferenciadas: el almacén, el refugio, la zona de instrucción, el comedor y la zona de descanso. Estos elementos son pasivos.

Las hormigas obreras se dividen en dos tipos recolectoras y reabastecedoras. Las recolectoras salen del hormiguero para buscar comida y cuando vuelven dejan la comida en el almacén. Las reabastecedoras cogen la comida del almacén y la llevan al comedor. Cada 10 iteraciones los dos tipos de hormiga paran a comer y descansar.

Las hormigas soldado se instruyen en la zona de instrucción y descansan, pero han de dejar lo que están haciendo cuando se aproxime un insecto invasor y repelerlo. Cada 6 iteraciones pararán para comer en el comedor.

Las hormigas crías comen y descansan, pero si llega un insecto invasor se refugiarán.

Todos los eventos que van sucediendo han de ser mostradas por pantalla en una interfaz gráfica, en la cual también habrá un botón que genere un insecto invasor y un botón que permita detener y reanudar todos los hilos.

Estos eventos también se escribirán en un registro.

También hay que añadir otro módulo responsable de conectarse de manera remota al servidor que ejecutará el comportamiento mencionado hasta ahora. Este módulo mostrará, de manera gráfica, estadísticas en tiempo real sobre el hormiguero, como el número de crías comiendo o el número de hormigas obreras fuera de la colonia. También dispondrá de un botón como el de la primera interfaz gráfica, capaz de generar un insecto invasor.

**Diseño general del sistema**

**Clases principales**

**Clases main:**

Servidor.java

La clase servidor, al iniciarse, crea una instancia del objeto operador, inicia el RMI registry en el puerto por defecto 1099 y publica el objeto remote op con una ruta de acceso predeterminada. Con esto inicia sus servicios como servidor. Después, crea un objeto VentanaPrincipal, lo coloca en medio de la pantalla y lo muestra. Se crea el actualizador de pantalla (objeto tipo ScreenUpdate) y se lanza. Tras eso, prepara el log para una nueva entrada, dejando una línea entre el log pasado y el nuevo log y mostrando que comienza una ejecución nueva. Finalmente, empieza a generar las hormigas, para generarlas según especifica el enunciado (“Cada 3 hormigas obreras creadas, se creará 1 hormiga soldado y 1 hormiga cría.”) se mira si el número de iteración del bucle es divisible entre 5, cabe destacar que el bucle empieza por la iteración 1 y no por la 0, si es divisible entre 5, es una hormiga soldado. Si no lo fuera, miraría si se corresponde con 1 módulo 5 (divide entre 5 y al resto le resta 1 y si ese número es 0 entonces es 1 módulo 5), si se corresponde entonces se tratará de una hormiga cría, en cualquier otro caso será una obrera.

Cliente.java

La clase cliente crea un objeto VentanaRemota, se explicará más adelante qué es, la posiciona en el centro de la pantalla y habilita su visibilidad. Tras eso crea un objeto tipo InterfazOperaciones, que se explicará en la sección “clases responsables de gestionar la conexión”, con el que se conecta al servidor. Después entra en un bucle infinito donde todo el rato está recibiendo los valores de los números que necesita para mostrarlos en los diferentes JTextFields, eso lo hace apoyándose en los métodos proporcionados por el objeto tipo InterfazOperaciones. Estos valores, como ya ha sido mencionado, se usan para actualizar los JTextFields de la clase VentanaRemota, haciendo uso de los métodos de la misma.

**Clases de hilos:**

Cria.java

Simula el comportamiento de las hormigas crías. Tiene cuatro atributos, uno de tipo String “nombre” otro de tipo entero “id”, otro de tipo ArrayList “listaCrias” y otro de tipo booleano “amenazado”. El ID se asigna según se van generando y con ese ID se crea el nombre para que quede con el formato HCXXXX donde las X se corresponden con el ID y si fuera un número menor que 1000, es decir, que no tuviera cuatro cifras, se rellenarán con ceros, como por ejemplo HC0007, esta hormiga tiene el ID 7. En el método run, primero se establece el name del Thread, no confundir con el atributo nombre, para que sea igual al mencionado, esto lo hacemos para que, por ejemplo al escribir en el log poder usar Threada.currentThread().getName() desde el método ejecutado sin tener que pasar como atributo el hilo que ejecuta dicho método. Nada más crearse crea un objeto Timestamp para poder registrar su nacimiento y crea un objeto de tipo TareaEscribir que está explicado posteriormente. Tras eso, le pasa dicha tarea al logger de la clase escritor para que lo escriba. Luego, hará en bucle lo siguiente: irá a comer y descansará, añadiéndose y quitándose de las listas del sitio donde está y deja de estar. Ajksdghaksjdghaklsjdhgaklsjdghakljsdhglkajeshgklaejrgha<wsejklrghaklejsdfghlakjhsdfgljkahdfgfjklahsdgkljahsdfgkljashdgjklaheg kajshdglkjashdgljkashdfgjkhasdljkghaklsdjhgalsjkdhglakjdsh kajlsdhfglajksdhgkjlgahsldkjghaksdjlhgaklj Si hubiera un ataque (función llamarAtaque()) el hilo se interrumpiría. En la gestión de la interrupción se llama a la función interrumpido(), en esta función la cría va al refugio y se añade a la lista de crías en el refugio, cuando salga, se quitará de dicha lista. Esta clase dispone además de métodos getter y setter para la variable “amenazado”. También tiene la función getNombre() que devuelve el nombre de la hormiga.

Hormiga.java

Esta es una interfaz que implementa Runnable, nos sirve para englobar a todos los tipos de hormiga que hay.

Obrera.java

Simula el comportamiento de las hormigas obreras. Tiene tres atributos, uno de tipo entero “iteración”, otro de tipo entero “id” y otro de tipo String “nombre”. El ID se asigna según se van generando y con ese ID se crea el nombre para que quede con el formato HOXXXX donde las X se corresponden con el ID y si fuera un número menor que 1000, es decir, que no tuviera cuatro cifras, el se rellenarán con ceros, como por ejemplo HO0077, esta hormiga tiene el ID 77. En el método run, primero se establece el name del Thread, no confundir con el atributo nombre, para que sea igual al mencionado, esto lo hacemos para que, por ejemplo al escribir en el log poder usar Threada.currentThread().getName() desde el método ejecutado sin tener que pasar como atributo el hilo que ejecuta dicho método. Nada más crearse crea un objeto Timestamp para poder registrar su nacimiento y crea un objeto de tipo TareaEscribir que está explicado posteriormente. Tras eso, le pasa dicha tarea al logger de la clase escritor para que lo escriba. A la hora de funcionar, las hormigas obreras se diferencian si su ID es par o si es impar. Si su ID es par repetirá en bucle el siguiente: irá al almacén, sacará comida del almacén, apuntará en el log que va a llevar comida al comedor, llevará la comida al comedor y la dejará ahí. Si su ID es impar: escribirá en el log que va a buscar comida, saldrá del hormiguero, buscará comida, entrará en el hormiguero, y dejará la comida en el almacén. Todas las hormigas cada 10 iteraciones de sus bucles paran a comer y descansar. También tiene la función getNombre() que devuelve el nombre de la hormiga.

Soldado.java

Simula el comportamiento de las hormigas soldado. Tiene 5 atributos, uno de tipo entero “id”, otro de tipo entero “iteración”, otro de tipo String “nombre”, otro de tipo CyclicBarrier “barrera” y otro de tipo CountDownLatch “latch”. El ID se asigna según se van generando y con ese ID se crea el nombre para que quede con el formato HSXXXX donde las X se corresponden con el ID y si fuera un número menor que 1000, es decir, que no tuviera cuatro cifras, se rellenarán con ceros, como por ejemplo HS0777, esta hormiga tiene el ID 777. En el método run, primero se establece el name del Thread, no confundir con el atributo nombre, para que sea igual al mencionado, esto lo hacemos para que, por ejemplo al escribir en el log poder usar Threada.currentThread().getName() desde el método ejecutado sin tener que pasar como atributo el hilo que ejecuta dicho método. Nada más crearse crea un objeto Timestamp para poder registrar su nacimiento y crea un objeto de tipo TareaEscribir que está explicado posteriormente. Tras eso, le pasa dicha tarea al logger de la clase escritor para que lo escriba. Luego entrará en un bucle en el que, se instruirá y descansará, y cada 6 iteraciones parará para comer. Hay una función llamarAtaque para simular cuando se genera un ataque, esta función interrumpe el hilo y crea una barrera y un countdownlatch. El hilo cuando se interrumpe llama a la función interrumpido. Esta función imprime en el log que la hormiga está defendiendo la colonia de la amenaza, luego el soldado saldrá de la colonia y esperará a que todos los soldados salgan del hormiguero. Cuando estén fuera todas empezará la pelea, cuando acabe el soldado volverá a entrar y seguirá haciendo su bucle. También tiene la función getNombre() que devuelve el nombre de la hormiga.

**Clases con variables compartidas:**

Almacen.java

Esta clase tiene 4 atributos, una de tipo entero “stock”, otra de tipo Semaphore “aforo”, otra de tipo Lock “control” y otra de tipo Condition “vacio”. La clase tiene tres métodos: incStock, decStock y getStock. incStock recibe como parámetro un entero, registra en el log que una hormiga ha entrado en el almacén para incrementar el stock y, si el semáforo lo permite, entra al almacén. Simula tardar de 2 a 5 segundos en dejar la comida y, de manera segura gracias al cerrojo “control” incrementa el stock disponible en el almacén inc unidades. Avisa a posibles hormigas reabastecedoras que estuvieran esperando de que ya hay stock disponible en el almacén. Finalmente sale del almacén. decStock recibe como parámetro un entero, registra en el log que una hormiga ha entrado en el almacén a decrementar el stock y, si el semáforo lo permite, entra al almacén. De manera protegida gracias a “control” accede a la variable stock, simula que durante 2-3 segundos coge la comida y decrementa el stock dec unidades. Si no pudiera decrementar el stock esperaría a que una hormiga reabastecedora aumentara el stock y la avisara. getStock() devuelve el stock que hay.

Bicho.java

Comedor.java

Esta clase tiene 3 atributos, uno de tipo entero “stock”, otra de tipo Lock “control” y otra de tipo Condition “vacio”. La clase tiene tres métodos: comer, incStock y getStock. La función comer recibe como parámetros dos enteros “comer” y “tiempo”, registra en el log que una hormiga está comiendo, simula que come de 2 a 3 segundos (esto se simula con un sleep) y, de manera segura gracias al cerrojo, decrementa el stock en comer unidades. Si no hubiera comida esperaría a que una hormiga reabastecedora mandara la señal de vacio, indicando que ya hay comida en el comedor. La función incStock recibe como parámetro un entero “inc”, registra en el log que una hormiga está incrementando el stock del comedor, deja la comida de 2 a 3 segundos (esto se simula con un sleep) y finalmente, de manera segura gracias al cerrojo, incrementa el stock inc unidades, enviando una señal de vacio a todas las hormigas que posiblemente estuvieran esperando a que hubiera comida para consumir. getStock() devuelve el stock que hay.

Descanso.java

Esta clase no tiene atributos. Tiene un método descansar que recibe como parámetro un entero “t”, registra en el log que una hormiga está descansando en el log y luego “descansa” t segundos, este descanso se simula con un sleep.

Hormiguero.java

Esta clase tiene 18 atributos que se corresponden con un atributo tipo Semaphore “salida”, otro tipo Lock “entrada”, cuatro de tipo entero “hormigasVivas”, “nObreras”, “nCriasComiendo” y “soldados”, otro tipo CyclicBarrier “barreraAtaque”, otro tipo CountDownLatch “bloqueoPelea” y finalmente 10 atributos tipo ArrayList, “almacen”, “comer”, “descanso”, “fuera”, “movimiento”, “dejandoComida”, “defendiendo”, “instrucd”, “refugio” y “listaSoldados”. También tiene los siguientes métodos: aumentarSoldados, entrar, salir, ataque y getters y setters para todas las listas y atributos previamente mencionados. aumentarSoldados es una función que no recibe parámetros e incrementa en uno el número de la variable “soldados”. entrar es una función que no recibe parámetros y, tras adquirir el cerrojo “entrada” simula su paso por ella con un sleep de 100ms y finalmente suelta el cerrojo, si no fuera posible adquirir el cerrojo entrada esperaría hasta que este quedara libre. salir es una función que no recibe parámetros y, tras adquirir el semáforo salida simula su paso por la salida con un sleep de 100ms y finalmente libera el semáforo, si no fuera posible adquirir el cerrojo esperaría hasta que este quedara libre. ataque es una función que no recibe parámetros de entrada, si hay más de un soldado, crea un CowntDownLatch “bloqueoPelea”, de tamaño 1 y una CyclicBarrier de tamaño soldados que cuando se libera, lanza un hilo tipo bicho con el CountDownLatch previamente creado, luego llama a la función llamarAtaque en soldado, con parámetros barreraAtaque y bloqueoPelea y también llama a la función llamarAtaque en Cría.

Instruc.java

Esta clase no tiene atributos. Tiene un método descansar que recibe como parámetro un entero “t”, registra en el log que una hormiga está descansando en el log y luego “descansa” t segundos, este descanso se simula con un sleep.

Refugio.java

Esta clase tiene 2 atributos, uno de tipo Lock “control” y otro de tipo Condition “espera”. Tiene dos funciones: refugiar y terminarAmenaza. refugiar es una función que no recibe parámetros, registra que la hormiga que ha llamado a esa función se ha metido al refugio. Tras eso, adquiere el lock de control y se pone a esperar la señal de espera para salir. terminarAmenaza establece la variable “amenazado” de las crías a false.

**Clases responsables de escribir en el fichero log**

Escritor.java

En esta clase se define un pool de hilos estático de tipo SingleThreadPool llamado “logger” que se encargará de gestionar todas las tareas de escritura en el log que se vayan generando conforme se vaya ejecutando el programa.

TareaEscribir.java

Esta clase, la cual implementa la interfaz Runnable crea las tareas que procesa el pool de un solo hilo mencionado previamente. Al método constructor se le pasan como atributos una cadena de caracteres que se corresponde con el nombre del hilo que ha generado la acción, un entero que se corresponde con el tipo de entrada del log que es y un objeto tipo Timestamp que nos ayudará a mostrar el momento en el que se genera la entrada. A partir del nombre del hilo se puede deducir el tipo de hormiga que está realizando la acción, simplemente hay que mirar el segundo carácter y a partir de él, si es una O entonces es Obrera, si es una C es Cría y si es S será soldado. En el run de esta tarea hay un simple método switch que dado el tipo de entrada que tiene que escribir en el log, el cual hemos pasado como parámetro, generará un mensaje u otro. Una vez se genera el mensaje se llama a la función toText, la cual recibe como parámetro una cadena de caracteres que en nuestro caso es el mensaje que ha generado la función run. toText crea un objeto tipo StringBuilder para poder formatear la cadena e introducir un salto de línea tras la entrada, para poder tener en cada línea del fichero una entrada y que no se solapen unas a otras. Una vez que se ha creado la cadena con el salto de línea se crea un objeto de tipo FileWriter al que le pasamos la dirección de memoria del archivo .txt donde queremos escribir los datos y de segundo parámetro true, para que en vez de sobreescribir el texto lo añada tras lo que había escrito. Escribe el mensaje y cierra el escritor.

**Clases responsables de mostrar las ventanas (JFrames)**

VentanaPrincipal.java

Esta clase es la encargada de mostrar la interfaz de usuario principal, la del servidor. Se crean los campos de texto, etiquetas y botones necesarios y se distribuyen de manera ordenada por la pantalla. Esta clase tiene el método updateData que se encarga de actualizar todos los campos que hay en el jFrame. Esto lo hace creando copias de las listas de hormigas que hay y recorriéndolas enseñando su contenido en los campos que hay. Botones funciones iuashgoaiuwrhgoiauwalo

VentanaRemota.java

Esta clase es la encargada de mostrar la interfaz del cliente que se conecta al servidor. Se crean los campos de texto, etiquetas y botón necesario y se distribuyen de manera ordenada por la pantalla. Esta clase tiene la función modificar, que recibe como parámetros un javax.swing.JTextField “campo” y un String “texto”. Lo que hace con esto es simplemente establecer el texto del JTextField a la cadena de caracteres que haya almacenada en texto. Boton funcioand oaishdguiaehrfughzu

**Clases responsables de gestionar la conexión:**

InterfazOperaciones.java

Interfaz remota en la que se encuentran definidos todos los métodos que estarán disponibles para el objeto remoto. Los métodos son: void generarAmenaza(), int getNObrerasFuera(), int getNObrerasDentro(), int getNSoldadosInstruc(), int getNSoldadosAmenaza(), int getNCriasComedor(), int getNCriasRefugio().

Operador.java

Responsable de la implementación de los métodos definidos en la interfaz remota “InterfazOperaciones”, sus métodos son los mismos que los de InterfazOperaciones, pero es aquí donde se encuentra la lógica de los mismos.   
generarAmenaza funciona igual que el generarAmenaza visto previamente en la clase hormiguero. El resto de métodos: getNObrerasFuera, getNObrerasdentero, getNSoldadosInstruc, getNSoldadosAmenaza, getNCriasComedor, getNCriasRefugio, devuelven un entero que es, en cada caso, el número de hilos que hay de un cierto tipo realizando una determinada acción, por ejemplo getNSoldadosAmenaza devuelve el número de soldados que hay defendiendo el hormiguero de una amenaza.

**Diagrama de clases**

**Código fuente**