Grado en Ingeniería Informática

Curso 2022/2023 – Convocatoria Ordinaria

03233102S – Gutiérrez Sánchez, Carlos

09087218X – Nieto Arias, Pablo

ÍNDICE

1) Análisis del problema

2) Diseño general del sistema

3) Descripción de las clases

4) diagrama de clases

5) Código fuente

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

La simulación de todos los tiempos se hace con sleeps. Todas las adquisiciones de cerrojo, tras hacer la acción especificada, como por ejemplo la modificación de una variable compartida, van seguidas de la liberación del mismo.

Para simular la entrada se ha usado un cerrojo y una función entrar que, tras adquirir el cerrojo, espera 100ms para simular el paso por la entrada y luego lo libera. Para simular la salida, en vez de dos túneles diferente de salida, se ha usado un semáforo. Se ha entendido como equivalente estos dos túneles por los que cabe 1 hormiga con un solo túnel donde caben 2 hormigas.

Para la asignación de IDs y nombres de las hormigas se han creado contadores para cada tipo de hormiga, así, conseguimos que cada tipo de hormiga tenga su hormiga número 0, hormiga número 1, … Para el nombre se hace una comprobación en el constructor de la hormiga: si la hormiga tiene un ID de una sola cifra el nombre será ST000X donde T es el tipo de hormiga que es y X es el ID, si tiene dos cifras el nombre será ST00XX donde T es el tipo de hormiga y X el ID, y así sucesivamente con tres y cuatro cifras.

A la hora de entrar al almacén, dependiendo del tipo de hormiga obrera del que se trate se hace una cosa u otra: las obreras impares, entran para reabastecer el almacén con comida que han traído de fuera de la colonia. Cuando llegan simulan dejar la comida y con un cerrojo acceden a la variable compartida que controla la cantidad de stock que queda y lo incrementan. Después mandarán un signalAll a las obreras pares de que hay stock. Esto lo hacemos porque si llega una obrera par al almacén y no queda, se saldrá y se quedará esperando a esta señal. Cuando estas entran hacen lo mismo que las pares, simulan coger la comida, obtienen el cerrojo y decrementan el stock.

En el comedor se hace algo muy similar, las obreras pares llevan la comida del almacén al comedor. Tras hacer el camino, de manera simulada, llegan al comedor, simula dejar la comida, obtiene el cerrojo que controla el acceso a esta variable compartida responsable de gestionar la cantidad de comida disponible en el comedor e incrementa dicha variable. Todas las hormigas, a la hora de comer hacen algo similar: acceden al cerrojo, decrementan la variable compartida del comedor, sueltan el cerrojo y luego simulan comer durante el tiempo que tengan que comer dependiendo del tipo de hormiga que sean.

En las zonas tanto de descanso como de instrucción no hay ningún tipo de variable compartida por lo que la función que simula la acción que se realiza en dicha acción solo es un sleep que simula el tiempo usado en realizarla.

Para todas estas zonas previamente mencionadas las hormigas se añadirán y quitarán a las listas correspondientes para seguir en qué punto del hormiguero se encuentran en cada momento.

Para el funcionamiento del botón de pausa se han envuelto todas las operaciones que realizan las hormigas en bloques try catch para que en cuanto se pulse el botón, se interrumpan todos los hilos. Esta interrupción se hace con un CountDownLatch de tamaño 1 que detiene a todas las hormigas y tras establecer una variable booleana a true llamada “pausa” espera a que se pulse de nuevo el botón, cuando se pulse de nuevo se pondrá a false “pausa” y se decrementará en 1 el CountDownLatch liberando satisfactoriamente los hilos. Cabe destacar que este botón no puede ser pulsado a la vez que hay una amenaza ya que ambas acciones se gestionan con excepciones y no son compatibles.

Para la generación de la amenaza y su funcionamiento se utiliza también un botón, que cuando se pulsa interrumpe a los soldados y a las crías, las obreras no verán su comportamiento afectado por la presencia de un insecto invasor. Para diferenciar de si la interrupción ha sido generada por el botón de pausa o por el de generar amenaza y gestionarlo adecuadamente, en la función que hay dentro del catch de las funciones de las hormigas miran la variable mencionada previamente "pausa" que se encuentra en Hormiguero, si está en false se trata de una amenaza, si está en true, se tratará del botón de pausa. Para las amenazas, las cuales solo podrán llegar cuando haya soldados vivos, se crea una CyclicBarrier que controle cuantos soldados tienen que ir a defender el hormiguero y una CountDownLatch, responsable de mantener a las crías en el refugio. Esta barrera también dispone de un objeto tipo Runnable que se lanzará cuando se llene la barrera que es la simulación del combate con el insecto invasor y el decremento del CountDownLatch para liberar a las crías del refugio. Cuando se crea la amenaza se llama a las funciones de llamarAtaque en cría y soldado que son las responsables de interrumpir estos hilos.

Para el funcionamiento del log, debido a que iba a haber muchas entradas simultáneas y proteger correctamente su acceso hemos creado un SingleThreadPool, de forma que, si solo hay un escritor no podrá haber condiciones de carrera. Para este pool, localizado en la clase Escritor.java se ha creado la clase TareaEscribir que implementa runnable y cada vez que hay una acción que hay que registrar como una hormiga comiendo o una hormiga instruyéndose se crea un objeto tipo TareaEscribir y se le pasa al pool para que lo ejecute. Para cada acción se ha definido de manera arbitraria códigos de escritura, por ejemplo, si a la función TareaEscribir se le pasa un 0 la acción es nacer, si se le pasan otros números las acciones serán diferentes. La TareaEscribir tiene la lógica necesaria para almacenar las entradas generadas en evoluciónColonia.txt

La conexión servidor-cliente ha sido realizada usando RMI. Se ha definido una interfaz "InterfazOperaciones" con métodos para generar amenazas y devolver valores númericos y se ha creado una clase "Operador" que implementa esta interfaz y contiene la lógica de estas funciones. En el servidor se ha creado un objeto tipo "Operador", se inicia el RMI en el puerto por defecto 1099 y se crea una ruta donde publica el objeto. En la clase cliente se instancia el objeto de tipo operador localizado en la ruta donde lo ha publicado el servidor, tras eso todas las operaciones definidas en la interfaz estarán disponibles para el cliente.

La VentanaPrincipal, responsable de mostrar por pantalla las acciones realizadas por las hormigas que hay creadas en el hormiguero dispone de los botones previamente mencionados de generación amenaza y pausa. Además, también contiene un método con el que actualizará sus campos de manera repetida. Para ello, dependiendo del campo se accederá a la lista que contiene las hormigas realizando dicha acción, creará una copia, para evitar problemas de buscar hormigas que ya han salido de dicha lista, y la recorrerá añadiéndola a una cadena que se establecerá como texto. Esta función es llamada por un hilo que se lanza a la vez y repite esta actualización de la pantalla de manera perpetua.

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 hilo 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. Antes de cada generación comprueba que el sistema no esté pausado, si lo estuviera, no generaría ninguna hormiga. Para generar las hormigas 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, al inicio de la ejecución 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 del método modificar (ver VentanaRemota.java. Si se pulsara el botón de generar amenaza, disponible en la interfaz remota, el cliente llamaría a la función amenaza, la cual crea el objeto tipo InterfazOperaciones y llama a la función generarAmenaza de esta interfaz lo cual genera en el hormiguero una amenaza.

Clases de hilos:

Cria.java

Esta clase implementa la interfaz Hormiga. Simula el comportamiento de las hormigas crías. Tiene tres atributos, uno de tipo String "nombre", otro de tipo ArrayList "listaCrias" y otro de tipo booleano "amenazado". El ID se pasa como parámetro al constructor 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, después de entrar en la colonia hará en bucle lo siguiente: irá a comer y descansará. 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, si el sistema no está pausado, manda a la cría al refugio, hasta que acabe la amenaza. 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, facilitando su seguimiento, pudiendo incluir objetos tipo "Hormiga" en listas.

Obrera.java

Esta clase implementa la interfaz Hormiga. Simula el comportamiento de las hormigas obreras. Tiene cuatro atributos, uno de tipo entero "iteración", otro de tipo entero "id", otro de tipo CountDownLatch "latch" 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. Su primera acción es entrar. 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. Se dispone de gestión de excepciones para cada método que se asegura de que no se pause el sistema, si se pausara el hilo se interrumpiría y se esperaría, gracias al CountDownLatch, a que se reanudara el sistema. También tiene la función getNombre que devuelve el nombre de la hormiga y setLatch que establece el CountDownLatch.

Soldado.java

Esta clase implementa la interfaz Hormiga. Simula el comportamiento de las hormigas soldado. Tiene 6 atributos, uno de tipo entero "id", otro de tipo entero "iteración", otro de tipo ArrayList "listaSoldados", 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, entra en el hormiguero y entonces 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, cuando interrumpe el hilo por una amenaza 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. Si la interrupción se debe al botón de pausa se hará lo mismo que se hacía con las otras hormigas. 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

Esta clase implementa Runnable. Tiene un atributo tipo CountDownLatch "pelea". En el constructor simplemente establece este latch como uno ya creado que se le pasa como parámetro. Su run tiene un sleep de 20 segundos, que simula la pelea con el insecto invasor, cuando acaba decrementa el latch liberándolo y avisa al refugio de que la lucha ha acabado, liberando a las crías.

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 un atributo tipo Semaphore “salida”, otro tipo Lock “entrada”, cuatro de tipo entero “hormigasVivas”, “nObreras”, “soldados” y “nCriasComiendo”, otro tipo CyclicBarrier “barreraAtaque”, otro tipo CountDownLatch “bloqueoPelea”, un CountDownLatch “latchPausa”, un booleano “pausa” y finalmente 9 atributos tipo ArrayList, “almacen”, “comer”, “descanso”, “fuera”, “movimiento”, “dejandoComida”, “defendiendo”, “instrucd” y “refugio”. También tiene los siguientes métodos: aumentarSoldados, entrar, salir, ataque, cambiaPausa 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. cambiaPausa comprueba la variable booleana pausa, si estuviera en true y se llamara a la función entonces se establecería a false dicha variable y decrementaría el latch, liberando a las hormigas. Si por el contrario está en false, crea un CountDownLatch de tamaño 1 y se lo pasa a todas las hormigas, establece a true la variable booleana y luego interrumpe a todas las hormigas.

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 en el log 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 manda la señal de salir a todas las hormigas crías que hubiera en el refugio.

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. El botón Generar Amenaza llama a la función de hormiguero ataque, la cual está explicada en la clase hormiguero. El botón cambiante de Pausar/reanudar cuando es pulsado llama a la función de hormiguero cambiaPausa y cambia el texto del botón de pausa a reanudar o viceversa.

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. El botón de generar amenaza llama a la función en cliente de amenaza que generará una amenaza en el servidor.

Clases responsables de gestionar la conexión:

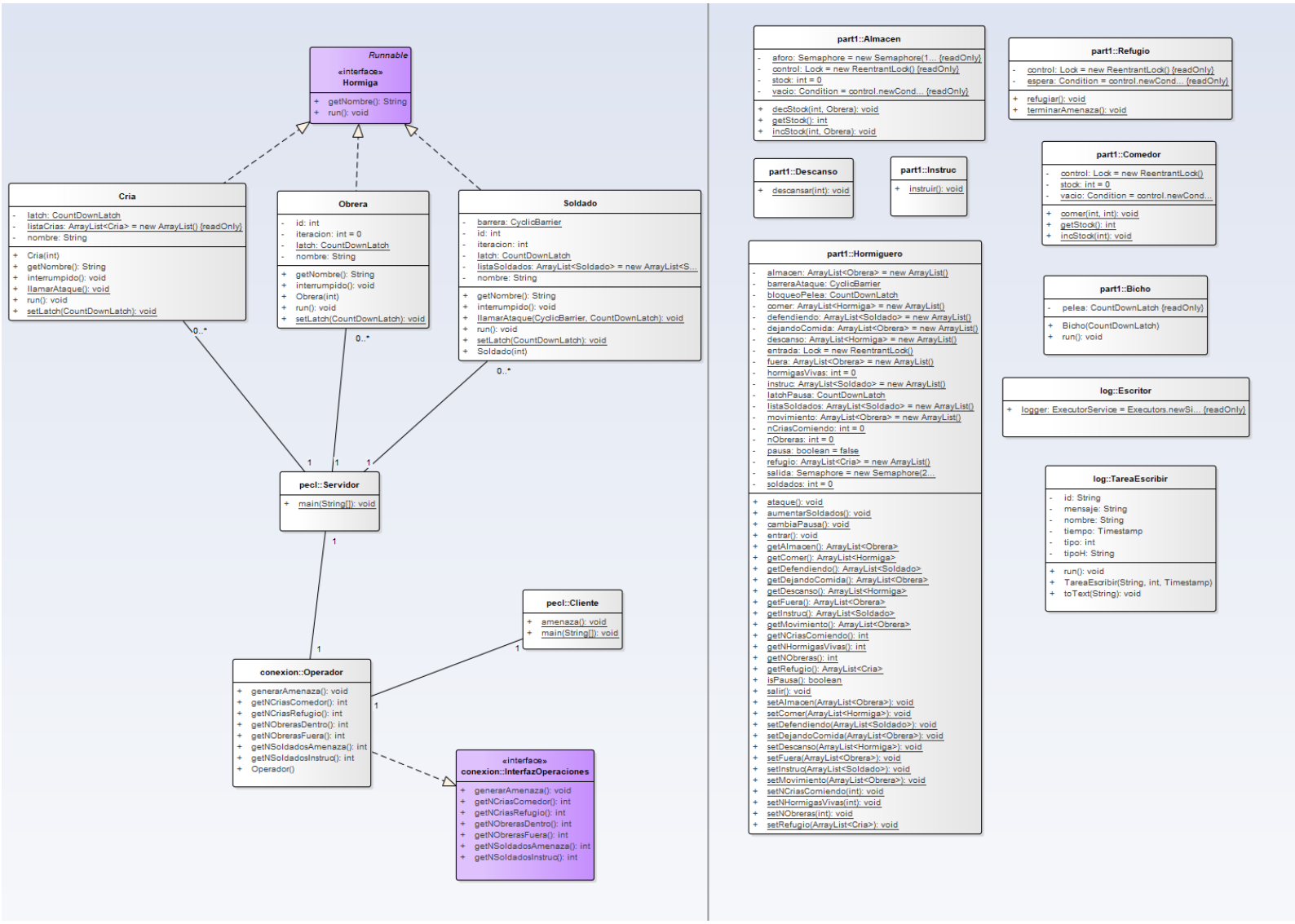
InterfazOperaciones.java

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

InterfazOperaciones.java:

```
package com.pablokarin.progav.conexion;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface InterfazOperaciones extends Remote
{
   //se declaran métodos para usar remotamente
   void generarAmenaza() throws RemoteException;
   int getNObrerasFuera() throws RemoteException;
   int getNObrerasDentro() throws RemoteException;
   int getNSoldadosInstruc() throws RemoteException;
   int getNSoldadosAmenaza() throws RemoteException;
   int getNCriasComedor() throws RemoteException;
   int getNCriasRefugio() throws RemoteException;

}
```

Operador.java

```
package com.pablokarin.progav.conexion;

import com.pablokarin.progav.part1.Hormiguero;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Operador extends UnicastRemoteObject implements InterfazOperaciones
{

   public Operador () throws RemoteException{}

   //genera amenaza en el hormiguero
```

```
public void generarAmenaza() throws RemoteException
{
   Hormiguero.ataque();
}

//devuelve el numero de Obreras fuera del hormiguero
public int getNObrerasFuera() throws RemoteException
{
   return Hormiguero.getFuera().size();
}

//devuelve el numero de Obreras dentro del hormiguero
public int getNObrerasDentro() throws RemoteException
{
   return Hormiguero.getNObreras()-Hormiguero.getFuera().size();
}

//devuelve el numero de Soldados instruyendose
public int getNSoldadosInstruc() throws RemoteException
{
   return Hormiguero.getInstruc().size();
}
//devuelve el numero de soldados defendiendo la amenaza
public int getNSoldadosAmenaza() throws RemoteException
{
   return Hormiguero.getDefendiendo().size();
}

//devuelve el numero de crias en el comedor
public int getNCriasComedor() throws RemoteException
{
   return Hormiguero.getNCriasComiendo();
```

```
    }

    //devuelve el numero de crias en el refugio
    public int getNCriasRefugio() throws RemoteException
    {
        return Hormiguero.getRefugio().size();
    }

}
```

VentanaPrincipal.java

```
package com.pablokarin.progav.jframe;

import java.awt.Color;
import com.pablokarin.progav.part1.*;
import com.pablokarin.progav.part1.hilos.*;
import java.util.ArrayList;

public class VentanaPrincipal extends javax.swing.JFrame {

    /**
     * Creates new form Frame
     */
    public VentanaPrincipal() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
```

```
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jScrollPane5 = new javax.swing.JScrollPane();
    jTree1 = new javax.swing.JTree();
    generarAmenaza = new javax.swing.JButton();
    pausaReanudar = new javax.swing.JToggleButton();
    jLabel1 = new javax.swing.JLabel();
    campoBuscandoComida = new javax.swing.JTextField();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    campoComiendo = new javax.swing.JTextArea();
    jScrollPane2 = new javax.swing.JScrollPane();
    campoDescansando = new javax.swing.JTextArea();
    jLabel7 = new javax.swing.JLabel();
    campoInstruyendose = new javax.swing.JTextField();
    campoComidaAlmacen = new javax.swing.JTextField();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();
    jLabel10 = new javax.swing.JLabel();
    campoComidaComedor = new javax.swing.JTextField();
    jScrollPane3 = new javax.swing.JScrollPane();
    campoDefendiendo = new javax.swing.JTextArea();
    jScrollPane4 = new javax.swing.JScrollPane();
    campoRefugio = new javax.swing.JTextArea();
    campoEnAlmacen = new javax.swing.JTextField();
    campoLlevandoComida = new javax.swing.JTextField();
```

```
        jLabel11 = new javax.swing.JLabel();
        campoDejandoComida = new javax.swing.JTextField();
        jLabel12 = new javax.swing.JLabel();
        campoNHormigas = new javax.swing.JTextField();

        jScrollPane5.setViewportView(jTree1);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Hormiguero");
        setIconImage(new javax.swing.ImageIcon("Other
sources/images/ormigabien.png").getImage());

        generarAmenaza.setBackground(new java.awt.Color(255, 0, 0));
        generarAmenaza.setForeground(new java.awt.Color(255, 255, 255));
        generarAmenaza.setText("Generar amenaza");
        generarAmenaza.setFocusable(false);
        generarAmenaza.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                generarAmenazaActionPerformed(evt);
            }
        });

        pausaReanudar.setText("Pausar");
        pausaReanudar.setFocusable(false);
        pausaReanudar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pausaReanudarActionPerformed(evt);
            }
        });

        jLabel1.setText("Hormigas buscando comida");

        campoBuscandoComida.setEditable(false);
```

```
campoBuscandoComida.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoBuscandoComidaActionPerformed(evt);
    }
});

jLabel2.setText("Hormigas comiendo");

jLabel3.setText("Hormigas descansando");

jLabel4.setText("Hormigas en el almacén");

jLabel5.setText("Comida en almacén:");

jLabel6.setText("Hormigas en el refugio");

campoComiendo.setLineWrap(true); // Habilitar el ajuste de línea
campoComiendo.setWrapStyleWord(true); // Ajustar palabras completas
campoComiendo.setEditable(false);
campoComiendo.setColumns(20);
campoComiendo.setRows(5);
jScrollPane1.setViewportView(campoComiendo);

campoDescansando.setEditable(false);
campoDescansando.setLineWrap(true); // Habilitar el ajuste de línea
campoDescansando.setWrapStyleWord(true); // Ajustar palabras completas
campoDescansando.setColumns(20);
campoDescansando.setRows(5);
jScrollPane2.setViewportView(campoDescansando);

jLabel7.setText("Hormigas defendiendo la colonia");
```

```
campoInstruyendose.setEditable(false);

campoComidaAlmacen.setEditable(false);
campoComidaAlmacen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoComidaAlmacenActionPerformed(evt);
    }
});

jLabel8.setText("Hormigas llevando comida");

jLabel9.setText("Hormigas instruyendose");

jLabel10.setText("Comida en comedor:");

campoComidaComedor.setEditable(false);
campoComidaComedor.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoComidaComedorActionPerformed(evt);
    }
});

campoDefendiendo.setLineWrap(true); // Habilitar el ajuste de línea
campoDefendiendo.setWrapStyleWord(true); // Ajustar palabras completas
campoDefendiendo.setEditable(false);
campoDefendiendo.setColumns(20);
campoDefendiendo.setRows(5);
jScrollPane3.setViewportView(campoDefendiendo);

campoRefugio.setLineWrap(true); // Habilitar el ajuste de línea
campoRefugio.setWrapStyleWord(true); // Ajustar palabras completas
campoRefugio.setEditable(false);
```

```
campoRefugio.setColumns(20);
campoRefugio.setRows(5);
jScrollPane4.setViewportView(campoRefugio);

campoEnAlmacen.setEditable(false);

campoLlevandoComida.setEditable(false);
campoLlevandoComida.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoLlevandoComidaActionPerformed(evt);
    }
});

jLabel11.setText("Hormigas dejando comida");

campoDejandoComida.setEditable(false);

jLabel12.setText("Hormigas vivas:");

campoNHormigas.setEditable(false);
campoNHormigas.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoNHormigasActionPerformed(evt);
    }
});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addGap(100, 100, 100)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 124,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jScrollPane1))
                    .addGap(45, 45, 45)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jScrollPane2)
                        .addComponent(jLabel3)))
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(pausaReanudar,
javax.swing.GroupLayout.PREFERRED_SIZE, 111,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel7)
                            .addComponent(jScrollPane3)))
                    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jScrollPane4)
                            .addComponent(jLabel6))
                        .addComponent(generarAmenaza)))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jLabel10)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(campoComidaComedor,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel5)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(campoComidaAlmacen,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel12)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(campoNHormigas,
javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 157, Short.MAX_VALUE))
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabel9, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabel11, javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel4, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jLabel8, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                .addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(campoDejandoComida,
javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(campoLlevandoComida,
javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(campoEnAlmacen)
                .addComponent(campoInstruyendose,
javax.swing.GroupLayout.Alignment.TRAILING)
                .addComponent(campoBuscandoComida))))
```

```
            .addGap(100, 100, 100))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGap(75, 75, 75)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(jLabel2))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 104,
Short.MAX_VALUE)
                    .addComponent(jScrollPane2))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel10)
                    .addComponent(campoComidaComedor,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel5)
                    .addComponent(campoComidaAlmacen,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel12)
                    .addComponent(campoNHormigas,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(13, 13, 13)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
```

```
                .addComponent(campoBuscandoComida,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel4)

                .addComponent(campoEnAlmacen,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel8)

                .addComponent(campoLlevandoComida,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel11)

                .addComponent(campoDejandoComida,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel9)

                .addComponent(campoInstruyendose,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                .addComponent(jLabel6)

                .addComponent(jLabel7))

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jScrollPane4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(20, 20, 20)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(generarAmenaza)
                .addComponent(pausaReanudar))
            .addGap(75, 75, 75))
        );

        pack();
    }// </editor-fold>

    private void pausaReanudarActionPerformed(java.awt.event.ActionEvent evt) {
        if (pausaReanudar.isSelected())
        {
            //se cambia el texto
            pausaReanudar.setText("Reanudar");
            //se establece el color a blanco
            //para que no cambie el color cuando se pulsa
            pausaReanudar.setBackground(Color.white);
            //se pausa el hormiguero
            Hormiguero.cambiaPausa();
        }
        if (!pausaReanudar.isSelected())
        {
            //cambia el texto
            pausaReanudar.setText("Pausar");
            //Se reanuda el hormiguero
            Hormiguero.cambiaPausa();
```

```
    }
}

private void generarAmenazaActionPerformed(java.awt.event.ActionEvent evt) {
    //genera un ataque en el hormiguero
    Hormiguero.ataque();
}

private void campoComidaComedorActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void campoBuscandoComidaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void campoComidaAlmacenActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void campoLlevandoComidaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void campoNHormigasActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
```

```
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Lev
el.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Lev
el.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Lev
el.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Lev
el.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
```

```
            new VentanaPrincipal().setVisible(true);
        }
    });
}

public void updateData()
{
    //establece los campos con los valores de las variables del hormiguero
    campoComidaAlmacen.setText(String.valueOf(Almacen.getStock()));
    campoComidaComedor.setText(String.valueOf(Comedor.getStock()));
    campoNHormigas.setText(String.valueOf(Hormiguero.getNHormigasVivas()));

    //para actualizar el almacén
    String almacen = "";
    //crea una copia de las hormigas en el almacen
    ArrayList<Obrera> lAlmacen = Hormiguero.getAlmacen();
    //recorre la lista copiada añadiendo a la
    //cadena las hormigas que hay
     for (int i = 0; i < lAlmacen.size(); i++)
     {
       try
       {
          almacen += lAlmacen.get(i).getNombre() + ", ";
       }
       catch(Exception e){}
     }
     //modifica el valor del campo
    campoEnAlmacen.setText(almacen);

    //para actualizar el comedor
    String comiendo = "";
    //crea una copia de las hormigas en el comedor
```

```
ArrayList<Hormiga> lComer = Hormiguero.getComer();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lComer.size(); i++)
{
   try
   {
      comiendo += lComer.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoComiendo.setText(comiendo);

//para actualizar zona de descanso
String descansando = "";
//cera una copia de las hormigas descansando
ArrayList<Hormiga> lDescanso = Hormiguero.getDescanso();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lDescanso.size(); i++)
{
   try
   {
      descansando += lDescanso.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoDescansando.setText(descansando);

//para actualizar fuera
```

```
String fuera = "";
//crea una copia de las hormigas fuera
ArrayList<Obrera> lFuera = Hormiguero.getFuera();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lFuera.size();i++)
{
   try
   {
      fuera += lFuera.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoBuscandoComida.setText(fuera);

//para actualizar hormigas moviéndose
String movimiento = "";
//crea una copia de las hormigas moviéndose con alimento
ArrayList<Obrera> lMovimiento = Hormiguero.getMovimiento();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lMovimiento.size(); i++)
{
   try
   {
      movimiento += lMovimiento.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoLlevandoComida.setText(movimiento);
```

```
//para actualizar hormigas dejando comida
String dejandoComida = "";
//crea una copia de las hormigas dejando comida
ArrayList<Obrera> lDejandoComida = Hormiguero.getDejandoComida();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lDejandoComida.size(); i++)
{
   try
   {
      dejandoComida += lDejandoComida.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoDejandoComida.setText(dejandoComida);

//para actualizar las hormigas defendiendo
String defendiendo = "";
//crea una copia de las hormigas defendiendo
ArrayList<Soldado> lDefendiendo = Hormiguero.getDefendiendo();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lDefendiendo.size(); i++)
{
   try
   {
      defendiendo += lDefendiendo.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
```

```
//modifica el valor del campo
campoDefendiendo.setText(defendiendo);

//para actualizar las hormigas instruyendose
String instruc = "";
//crea una copia de las hormigas instruyendose
ArrayList<Soldado> lInstruc = Hormiguero.getInstruc();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lInstruc.size(); i++)
{
   try
   {
      instruc += lInstruc.get(i).getNombre() + ", ";
   }
   catch(Exception e){}
}
//modifica el valor del campo
campoInstruyendose.setText(instruc);

//para atualizar las crías refugiadas
String refugio = "";
//crea una copia de las hormigas refugiadas
ArrayList<Cria> lRefugio = Hormiguero.getRefugio();
//recorre la lista copiada añadiendo a la
//cadena las hormigas que hay
for (int i = 0; i < lRefugio.size(); i++)
{
   try
   {
      refugio += lRefugio.get(i).getNombre() + ", ";
   }
```

```
        catch(Exception e){}
    }
    //modifica el valor del campo
    campoRefugio.setText(refugio);
}



// Variables declaration - do not modify
private javax.swing.JTextField campoBuscandoComida;
private javax.swing.JTextField campoComidaAlmacen;
private javax.swing.JTextField campoComidaComedor;
private javax.swing.JTextArea campoComiendo;
private javax.swing.JTextArea campoDefendiendo;
private javax.swing.JTextField campoDejandoComida;
private javax.swing.JTextArea campoDescansando;
private javax.swing.JTextField campoEnAlmacen;
private javax.swing.JTextField campoInstruyendose;
private javax.swing.JTextField campoLlevandoComida;
private javax.swing.JTextField campoNHormigas;
private javax.swing.JTextArea campoRefugio;
private javax.swing.JButton generarAmenaza;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
```

```
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JTree jTree1;
    private javax.swing.JToggleButton pausaReanudar;
    // End of variables declaration
}
```

VenataRemota.java

```
package com.pablokarin.progav.jframe;

import com.pablokarin.progav.pecl.Cliente;
import javax.swing.JTextField;



public class VentanaRemota extends javax.swing.JFrame {

    /**
     * Creates new form VentanaRemota
     */
    public VentanaRemota() {
        initComponents();
    }

    public JTextField getCampoNCriasComedor() {
        return campoNCriasComedor;
```

```
}

public JTextField getCampoNCriasRefugio() {
    return campoNCriasRefugio;
}

public JTextField getCampoNObrerasDentro() {
    return campoNObrerasDentro;
}

public JTextField getCampoNObrerasFuera() {
    return campoNObrerasFuera;
}

public JTextField getCampoNSoldadosAmenaza() {
    return campoNSoldadosAmenaza;
}

public JTextField getCampoNSoldadosInstruc() {
    return campoNSoldadosInstruc;
}



/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
```

```
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        campoNObrerasFuera = new javax.swing.JTextField();
        campoNObrerasDentro = new javax.swing.JTextField();
        campoNSoldadosInstruc = new javax.swing.JTextField();
        campoNSoldadosAmenaza = new javax.swing.JTextField();
        campoNCriasComedor = new javax.swing.JTextField();
        campoNCriasRefugio = new javax.swing.JTextField();
        generarAmenaza = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setIconImage(new javax.swing.ImageIcon("Other
sources/images/ormiga").getImage());

        jLabel2.setText("Número de hormigas obreras fuera del hormiguero:");

        jLabel3.setText("Número de hormigas obreras dentro del hormiguero:");

        jLabel4.setText("Número de hormigas soldado instruyéndose:");

        jLabel5.setText("Número de hormigas soldado repeliendo una amenaza:");

        jLabel6.setText("Número de hormigas cría en el comedor:");
```

```
jLabel7.setText("Número de hormigas cría en el refugio:");

campoNObrerasFuera.setEditable(false);

campoNObrerasDentro.setEditable(false);

campoNSoldadosInstruc.setEditable(false);
campoNSoldadosInstruc.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoNSoldadosInstrucActionPerformed(evt);
    }
});

campoNSoldadosAmenaza.setEditable(false);

campoNCriasComedor.setEditable(false);
campoNCriasComedor.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoNCriasComedorActionPerformed(evt);
    }
});

campoNCriasRefugio.setEditable(false);
campoNCriasRefugio.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        campoNCriasRefugioActionPerformed(evt);
    }
});
```

```
        generarAmenaza.setBackground(new java.awt.Color(255, 0, 0));
        generarAmenaza.setForeground(new java.awt.Color(255, 255, 255));
        generarAmenaza.setText("Generar Amenaza");
        generarAmenaza.setFocusable(false);
        generarAmenaza.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                generarAmenazaStateChanged(evt);
            }
        });
        generarAmenaza.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                generarAmenazaActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap(44, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addComponent(jLabel3)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G, false)
                        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                            .addComponent(jLabel6)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                    .addComponent(campoNCriasComedor,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))

                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()


.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
, false)

                    .addComponent(jLabel5,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

                        .addComponent(jLabel7,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                        .addGap(77, 77, 77)))


.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
, false)

                    .addGroup(layout.createSequentialGroup()


.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

                        .addComponent(campoNSoldadosAmenaza,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))

                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()


.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                    .addComponent(campoNCriasRefugio,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))))

                .addGroup(layout.createSequentialGroup()
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addComponent(jLabel4)
                    .addComponent(jLabel2))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addComponent(campoNObrerasFuera,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(campoNSoldadosInstruc,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(campoNObrerasDentro,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGroup(layout.createSequentialGroup()
                    .addGap(126, 126, 126)
                    .addComponent(generarAmenaza)))
            .addContainerGap(44, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap(35, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                    .addComponent(jLabel2)
                    .addComponent(campoNObrerasFuera,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel3)
                .addComponent(campoNObrerasDentro,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel4)
                .addComponent(campoNSoldadosInstruc,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel5)
                .addComponent(campoNSoldadosAmenaza,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel6)
```

```
                .addComponent(campoNCriasComedor,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel7)
                .addComponent(campoNCriasRefugio,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addComponent(generarAmenaza)
            .addContainerGap(48, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void campoNCriasComedorActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
    }

    private void campoNSoldadosInstrucActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
    }

    private void campoNCriasRefugioActionPerformed(java.awt.event.ActionEvent evt)
{
        // TODO add your handling code here:
```

```
    }

    private void generarAmenazaActionPerformed(java.awt.event.ActionEvent evt) {
        //llama a la función amenaza en cliente
        Cliente.amenaza();
    }

    private void generarAmenazaStateChanged(javax.swing.event.ChangeEvent evt) {
        // TODO add your handling code here:
    }

    /**
     * @param args the command line arguments
     */
    public void modificar (javax.swing.JTextField campo, String texto)
    {
        //modifica el campo
        campo.setText(texto);
    }
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
```

```
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
java.util.logging.Logger.getLogger(VentanaRemota.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
java.util.logging.Logger.getLogger(VentanaRemota.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
java.util.logging.Logger.getLogger(VentanaRemota.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
java.util.logging.Logger.getLogger(VentanaRemota.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new VentanaRemota().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JTextField campoNCriasComedor;
```

```
    private javax.swing.JTextField campoNCriasRefugio;
    private javax.swing.JTextField campoNObrerasDentro;
    private javax.swing.JTextField campoNObrerasFuera;
    private javax.swing.JTextField campoNSoldadosAmenaza;
    private javax.swing.JTextField campoNSoldadosInstruc;
    private javax.swing.JButton generarAmenaza;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    // End of variables declaration
}
```

Escritor.java

```
package com.pablokarin.progav.log;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Escritor {
    //se crea el pool de un solo hilo que gestiona la escritura en log
    public static final ExecutorService logger = Executors.newSingleThreadExecutor();

}
```

TareaEscribir.java

```
package com.pablokarin.progav.log;

import java.io.FileWriter;
import java.io.IOException;
```

```
import java.sql.Timestamp;

public class TareaEscribir implements Runnable{
    private String nombre;
    private String id;
    private String tipoH; //tipo de hormiga (obrera, soldado o cría)
    private int tipo; //tipo de escritura que se va a realizar
    private Timestamp tiempo;
    private String mensaje; //lo que se escribe en el log

    public TareaEscribir(String nombre, int tipo, Timestamp tiempo)
    {
        this.nombre = nombre;
        //dependiendo del segundo caracter lo hará una hormiga u otra
        switch(nombre.charAt(1))
        {
            case 'O':
            {
                tipoH = "Obrera";
                break;
            }
            case 'S':
            {
                tipoH = "Soldado";
                break;
            }
            case 'C':
            {
                tipoH = "Cría";
                break;
            }
```

```
        }
        this.tipo = tipo;
        this.tiempo = tiempo;
    }

    public void run()
    {
        switch (tipo)
        {
            case 0://nacer
            {
                mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", ha nacido.";
                break;
            }
            case 1: //comer
            {
                mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está comiendo.";
                break;
            }
            case 2://Descansar
            {
                mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está descansando.";
                break;
            }
            case 3: //Recoger comida
            {
                mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está recogiendo
comida.";
                break;
            }
            case 4://Guardar comida
            {
```

```
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está guardando comida en el almacén.";
            break;
        }
        case 5://Sacar comida
        {
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está sacando comida del almacén.";
            break;
        }
        case 6: //Llevar comida
        {
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está llevando comida del almacén al comedor.";
            break;
        }
        case 7://Dejar comida
        {
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está dejando comida en el comedor.";
            break;
        }
        case 8://Instruirse
        {
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está instruyéndose.";
            break;
        }
        case 9://Defender
        {
            mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", va a defender a la colonia de una amenaza.";
            break;
        }
        case 10://Refugiada
```

```
            {
                mensaje = tiempo + " : la hormiga " + tipoH + ", " + nombre + ", está refugiada.";
                break;
            }

        }
        toText(mensaje);
    }

    public void toText(String mensaje)
    {
        //crea la cadena con un salto de línea
        StringBuilder sb = new StringBuilder();
        sb.append(mensaje);
        sb.append((System.lineSeparator()));
        mensaje = sb.toString();
        try
        {
            //localiza el archivo donde escribir
            //parámetro true para añadir lo escrito al final del log
            FileWriter fileWriter = new
FileWriter("src/main/java/com/pablokarin/progav/log/evolucionColonia.txt", true);
            //Escribe el string mensaje en el documento de texto
            fileWriter.write(mensaje);
            //cierra el escritor
            fileWriter.close();
        }
        catch (IOException e) {System.out.println("No se pudo escribir en el archivo. Error de I/0:
" + e);}

    }
}
```

Almacen.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import com.pablokarin.progav.part1.hilos.Obrera;
import java.sql.Timestamp;
import java.util.concurrent.Semaphore;
import java.util.Random;
import java.util.concurrent.locks.*;

public class Almacen
{
   private static int stock = 0;
   private static final Semaphore aforo = new Semaphore(10, true);
   private static final Lock control = new ReentrantLock();
   private static final Condition vacio = control.newCondition();

   //incrementa el stock del almacen
   public static void incStock(int inc, Obrera obr) throws InterruptedException
   {
      Timestamp timestamp = new Timestamp(System.currentTimeMillis());
      TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 4,
timestamp);
      Escritor.logger.execute(entrada);

      try
      {
         //entra al almacen
         aforo.acquire();
         //gestion de listas
         Hormiguero.getAlmacen().add(obr);
         //descarga
         Thread.sleep((new Random().nextInt(3) + 2)*1000);
```

```
            //obtiene el lock de control
            control.lock();
            //modifica la variable
            stock += inc;
            //avisa a los que estan esperando por stock
            vacio.signalAll();
        }
        finally
        {
            try
            {
            control.unlock();
            }
            catch (Exception e) {}
            Hormiguero.getAlmacen().remove(obr);
            aforo.release();
        }
    }

    public static int getStock() {
        return stock;
    }

    //disminuye el stock del almacen
    public static synchronized void decStock(int dec, Obrera obr) throws InterruptedException
    {
        Timestamp timestamp = new Timestamp(System.currentTimeMillis());
        TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 5,
timestamp);
        Escritor.logger.execute(entrada);

        try
        {
```

```
    //entra al almacen
    aforo.acquire();
    //gestion de listas
    Hormiguero.getAlmacen().add(obr);
    control.lock();
    //mira la variable y si no hay stock se sale del almacen a esperar
    while (stock < dec)
    {
        Hormiguero.getAlmacen().remove(obr);

        //para evitar que se quede el almacen lleno de hormigas esperando por stock
        aforo.release();

        //sigue desde el await cuando ocurre el signal
        vacio.await();

        //vuelve a entrar al almacen
        aforo.acquire();

        Hormiguero.getAlmacen().add(obr);
    }
    //saca comida
    Thread.sleep((new Random().nextInt(2) + 1) * 1000);
    stock -= dec;

}
finally
{
    try
    {
        control.unlock();
    }
```

```
            catch(Exception e)
            {
               System.out.println(e.getMessage());
            }
            try
            {
               Hormiguero.getAlmacen().remove(obr);
               aforo.release();
            }
            catch(Exception e)
            {
               System.out.println(e.getMessage());
            }
         }
      }
   }
```

Bicho.java

```
package com.pablokarin.progav.part1;

import static java.lang.Thread.sleep;
import java.util.concurrent.CountDownLatch;

public class Bicho implements Runnable
{
   private final CountDownLatch pelea;

   public Bicho(CountDownLatch l)
   {
      pelea = l;
   }

   @Override
```

```
    public void run()
    {
        //comienza la pelea (dura 20 s)
        try
        {
            sleep(20000);
        }
        catch(InterruptedException IE){}

        //al terminar la pelea libera a todas las soldado peleando
        pelea.countDown();

        //notifica a las crias de que la amenaza ha terminado
        System.out.println("Voy a notificar");
        Refugio.terminarAmenaza();
    }
}
```

Comedor.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import java.sql.Timestamp;
import java.util.Random;
import java.util.concurrent.locks.*;

public class Comedor
{
    private static int stock = 0;
    private static final Lock control = new ReentrantLock();
    private static final Condition vacio = control.newCondition();
```

```
    public static int getStock()
    {
        return stock;
    }
    //pilla del stock y luego come
    public static void comer(int comer, int tiempo) throws InterruptedException
    {

        Timestamp timestamp = new Timestamp(System.currentTimeMillis());
        TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 1,
timestamp);
        Escritor.logger.execute(entrada);

        //prueba a comer
        try
        {
            control.lock();
            //mira si hay stock y si no espera
            while (stock == 0)
            {
                vacio.await();
            }
            //come
            stock -= comer;

        }
        finally
        {
            control.unlock();
        }

        Thread.sleep(tiempo * 1000);
    }
```

```
    public static void incStock(int inc) throws InterruptedException
    {
        Timestamp timestamp = new Timestamp(System.currentTimeMillis());
        TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 7,
timestamp);
        Escritor.logger.execute(entrada);

        //incrementa el stock del comedor
        Thread.currentThread().sleep((new Random().nextInt(2) + 1) * 1000);

        try
        {
            control.lock();
            stock += inc;
            vacio.signalAll();
        }
        finally
        {
            control.unlock();
        }
    }

}
```

Descanso.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import java.sql.Timestamp;

public class Descanso {
    public static void descansar(int t) throws InterruptedException
```

```
    {
        Timestamp timestamp = new Timestamp(System.currentTimeMillis());
        TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 2,
timestamp);
        Escritor.logger.execute(entrada);

        //duerme t segundos
        Thread.sleep(t*1000);
    }
}
```

Hormiguero.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.part1.hilos.*;
import java.util.ArrayList;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.CountDownLatch;



public class Hormiguero
{
    //control para entradas y salidas
    private static final Semaphore salida = new Semaphore(2,true);
    private static final Lock entrada = new ReentrantLock();

    //contadores de hormigas
    private static int hormigasVivas = 0;
    private static int nObreras = 0;
    private static int nCriasComiendo = 0;
```

```
//bloqueos para la amenaza
private static CyclicBarrier barreraAtaque;
private static CountDownLatch bloqueoPelea;

//bloqueos para la pausa
private static CountDownLatch latchPausa;

private static boolean pausa = false;
public static boolean isPausa()
{
   return pausa;
}

//listas de hormigas para ScreenUpdate
private static ArrayList<Obrera> almacen = new ArrayList();
private static ArrayList<Hormiga> comer = new ArrayList();
private static ArrayList<Hormiga> descanso = new ArrayList();
private static ArrayList<Obrera> fuera = new ArrayList();
private static ArrayList<Obrera> movimiento = new ArrayList();
private static ArrayList<Obrera> dejandoComida = new ArrayList();
private static ArrayList<Soldado> defendiendo = new ArrayList();
private static ArrayList<Soldado> instruc = new ArrayList();
private static ArrayList<Cria> refugio = new ArrayList();

//cantidad de soldados
private static int soldados = 0;
//aumenta la cantidad de soldados
public static void aumentarSoldados() {
   soldados++;
}

//entrar a la colonia
```

```
public static void entrar()throws InterruptedException
{
   try
   {
      entrada.lock();
      Thread.sleep(100);
   }

   finally
   {
      entrada.unlock();
   }
}

//salir de la colonia
public static void salir() throws InterruptedException
{
   try
   {
      salida.acquire();
      Thread.sleep(100);
   }

   finally
   {
      salida.release();
   }
}
//se llama al iniciar un ataque
public static void ataque()
{
   if(soldados > 0)
```

```
    {
        //inicializa los bloqueos
        bloqueoPelea = new CountDownLatch(1);
        barreraAtaque = new CyclicBarrier( soldados, new Bicho(bloqueoPelea));

        //avisa a las soldado
        Soldado.llamarAtaque(barreraAtaque, bloqueoPelea);

        //avisa a las crias
        Cria.llamarAtaque();
    }
}

//logica del boton de pausa
public static void cambiaPausa()
{
    //se rige por un CountDownLatch
    if(!pausa)
    {
        latchPausa = new CountDownLatch(1);

        //referencia el latch a las hormigas
        Obrera.setLatch(latchPausa);
        Soldado.setLatch(latchPausa);
        Cria.setLatch(latchPausa);

        //pausa la simulacion
        pausa = true;

        //mira entre todas la threads de la ejecucion
        for (Thread t : Thread.getAllStackTraces().keySet())
        {
```

```
            //pilla las que son hormigas
            if (t.getName().contains("HO") || t.getName().contains("HS") ||
t.getName().contains("HC"))
            {
               //System.out.println(t.getName());
               t.interrupt();
            }
         }
      }
      else
      {
         //despausa
         pausa = false;
         //decrementa el latch para liberarlo
         latchPausa.countDown();
      }

   }

   // <editor-fold desc="GETTER AND SETTER">
   public synchronized static int getNObreras()
   {
      return nObreras;
   }
   public static void setNObreras(int nObreras)
   {
      Hormiguero.nObreras = nObreras;
   }
   public synchronized static int getNHormigasVivas()
   {
      return hormigasVivas;
   }
   public static void setNHormigasVivas(int hormigasVivas)
```

```
{
    Hormiguero.hormigasVivas = hormigasVivas;
}

public synchronized static int getNCriasComiendo()
{
    return nCriasComiendo;
}

public static void setNCriasComiendo(int nCriasComiendo)
{
    Hormiguero.nCriasComiendo = nCriasComiendo;
}

public synchronized static ArrayList<Obrera> getAlmacen() {
    return almacen;
}

public static void setAlmacen(ArrayList<Obrera> almacen) {
    Hormiguero.almacen = almacen;
}

public synchronized static ArrayList<Hormiga> getComer() {
    return comer;
}

public static void setComer(ArrayList<Hormiga> comer) {
    Hormiguero.comer = comer;
}

public synchronized static ArrayList<Hormiga> getDescanso() {
    return descanso;
```

```
}

public static void setDescanso(ArrayList<Hormiga> descanso) {
   Hormiguero.descanso = descanso;
}

public synchronized static ArrayList<Obrera> getFuera() {
   return fuera;
}

public static void setFuera(ArrayList<Obrera> fuera) {
   Hormiguero.fuera = fuera;
}

public synchronized static ArrayList<Obrera> getMovimiento() {
   return movimiento;
}

public static void setMovimiento(ArrayList<Obrera> movimiento) {
   Hormiguero.movimiento = movimiento;
}

public synchronized static ArrayList<Obrera> getDejandoComida()
{
   return dejandoComida;
}

public static void setDejandoComida(ArrayList<Obrera> dejandoComida)
{
   Hormiguero.dejandoComida = dejandoComida;
}
```

```
    public synchronized static ArrayList<Soldado> getDefendiendo() {
        return defendiendo;
    }

    public static void setDefendiendo(ArrayList<Soldado> defendiendo) {
        Hormiguero.defendiendo = defendiendo;
    }

    public synchronized static ArrayList<Soldado> getInstruc() {
        return instruc;
    }

    public static void setInstruc(ArrayList<Soldado> instruc) {
        Hormiguero.instruc = instruc;
    }

    public synchronized static ArrayList<Cria> getRefugio() {
        return refugio;
    }

    public static void setRefugio(ArrayList<Cria> refugio) {
        Hormiguero.refugio = refugio;
    }
    // </editor-fold>
}
```

Instruc.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import java.sql.Timestamp;
import java.util.Random;
```

```
public class Instruc {
   public static void instruir() throws InterruptedException
   {
      Timestamp timestamp = new Timestamp(System.currentTimeMillis());
      TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 8,
timestamp);
      Escritor.logger.execute(entrada);

      //gastan tiempo en instruirse
      Thread.sleep((new Random().nextInt(7) + 2) * 1000);
   }
}
```

Refugio.java

```
package com.pablokarin.progav.part1;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import java.sql.Timestamp;
import java.util.concurrent.locks.*;

public class Refugio {

   private static final Lock control = new ReentrantLock();
   private static final Condition espera = control.newCondition();

   //mete a las crias en el refugio
   public static void refugiar() throws InterruptedException
   {
      Timestamp timestamp = new Timestamp(System.currentTimeMillis());
      TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 10,
timestamp);
      Escritor.logger.execute(entrada);
```

```
        try
        {
            control.lock();
            espera.await();
        }
        finally
        {
            control.unlock();
        }
    }

    //avisa de que tienen que salir del refugio
    public static void terminarAmenaza()
    {
        try
        {
            control.lock();
            espera.signalAll();
        }
        finally
        {
            control.unlock();
        }
    }
}
```

Cria.java

```
package com.pablokarin.progav.part1.hilos;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import com.pablokarin.progav.part1.*;
import java.sql.Timestamp;
```

```
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.CountDownLatch;



public class Cria implements Hormiga
{

   private String nombre;
   //lista estatica de crias
   private static final ArrayList<Cria> listaCrias = new ArrayList();
   private static CountDownLatch latch;

   public Cria (int id)
   {
      //agrega a lista estatica de crias
      listaCrias.add(this);

      //genera el nombre de la hormiga
      if (id < 10)
      {
         nombre = "HC000" + id;
      }
      else
      {
         if (id < 100)
         {
            nombre = "HC00" + id;
         }
         else
         {
            if (id<1000)
```

```
            {
                nombre = "HC0" + id;
            }
            else
            {
                if (id<10000)
                {
                    nombre = "HC" + id;
                }
            }
        }
    }
}

@Override
public String getNombre()
{
    return nombre;
}

public void run()
{
    //pone nombre al hilo
    Thread.currentThread().setName(nombre);

    //Aumenta el numero de hormigas vivas
    Hormiguero.setNHormigasVivas(Hormiguero.getNHormigasVivas()+1);

    //timestamps para el logger
    Timestamp timestamp1 = new Timestamp(System.currentTimeMillis());
    TareaEscribir entrada1 = new TareaEscribir(Thread.currentThread().getName(), 0,
timestamp1);
    Escritor.logger.execute(entrada1);
```

```
//entra recien nacida
try
{
   Hormiguero.entrar();
}
catch (InterruptedException ex)
{
   interrumpido();
}

//comienza el comportamiento
while (true)
{
   //intenta dormir
   try
   {
      //actualiza las listas y come
      Hormiguero.getComer().add(this);
      Hormiguero.setNCriasComiendo(Hormiguero.getNCriasComiendo()+1);
      Comedor.comer(1, new Random().nextInt(2) + 3);
      Hormiguero.setNCriasComiendo(Hormiguero.getNCriasComiendo()-1);
      Hormiguero.getComer().remove(this);
   }
   catch(InterruptedException IE)
   {
      //gestión de listas en interrupciones
      boolean pausado = false;
      if (!Hormiguero.isPausa())
      {
         Hormiguero.getComer().remove(this);
      }
```

```
    else
    {
        pausado = true;
    }

    //ocurre una interrupción
    interrumpido();

    if (pausado)
    {
        Hormiguero.getComer().remove(this);
    }
}

//intenta descansar
try
{
    //actualiza las listas y descansa
    Hormiguero.getDescanso().add(this);
    Descanso.descansar(4);
    Hormiguero.getDescanso().remove(this);
}
catch(InterruptedException IE)
{
    //gestion de listas
    boolean pausado = false;
    if (!Hormiguero.isPausa())
    {
        Hormiguero.getDescanso().remove(this);
    }
    else
    {
```

```
            pausado = true;
        }

        //gestiona la interrupcion
        interrumpido();

        if(pausado)
        {
            Hormiguero.getDescanso().remove(this);
        }
    }
  }
}

//protocolo en caso de ataque
public static void llamarAtaque()
{
  ArrayList<Thread> crias = new ArrayList();

  //toma todas las crias nacidas hasta el momento y
  // las interrumpe
  for(int i = 0; i < listaCrias.size(); i++)
  {
    String nombre = listaCrias.get(i).getNombre();
    //busca entre todos los hilos del programa
    for (Thread t : Thread.getAllStackTraces().keySet())
    {
      //si tiene nombre como la cria
      if(t.getName().equals(nombre))
      {
        //interrumpe
        t.interrupt();
```

```
            }
        }
    }
}

//protocolo a seguir en caso de interrupción
public void interrumpido()
{
    //distingue entre pausa y amenaza
    if (Hormiguero.isPausa())
    {
        //entra al CountDownLatch de la pausa
        try {
            latch.await();
        } catch (InterruptedException ex)
        {
            //por si acaso
            interrumpido();
        }
    }
    else
    {
        try
        {
            //actualiza las listas y entra al refugio
            Hormiguero.getRefugio().add(this);

            Refugio.refugiar();

            Hormiguero.getRefugio().remove(this);
        }
        catch(InterruptedException IE)
```

```
            {
                //por si acaso
                interrumpido();
            }
        }
    }

    public static void setLatch(CountDownLatch latch) {
        Cria.latch = latch;
    }
}
```

Hormiga.java

```
package com.pablokarin.progav.part1.hilos;

public interface Hormiga extends Runnable
{
    public void run();
    public String getNombre();



}
```

Obrera.java

```
package com.pablokarin.progav.part1.hilos;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import com.pablokarin.progav.part1.*;
import java.sql.Timestamp;
import java.util.Random;
import java.util.concurrent.CountDownLatch;

public class Obrera implements Hormiga{
```

```
private int iteracion = 0;
private final int id;
private String nombre;
private static CountDownLatch latch;

public static void setLatch(CountDownLatch latch) {
    Obrera.latch = latch;
}

public Obrera (int id)
{
   this.id = id;
   if (id < 10)
    {
       nombre = "HO000" + id;
    }
    else
    {
       if (id < 100)
       {
          nombre = "HO00" + id;
       }
       else
       {
          if (id<1000)
          {
             nombre = "HO0" + id;
          }
          else
          {
             if (id<10000)
             {
```

```
                    nombre = "HO" + id;
                }
            }
        }
    }

    @Override
    public String getNombre()
    {
        return nombre;
    }

    @Override
    public void run()
    {
        //pone nombre al hilo
        Thread.currentThread().setName(nombre);

        //actualiza el contador de hormigas vivas
        Hormiguero.setNHormigasVivas(Hormiguero.getNHormigasVivas()+1);
        //actualiza el contador de obreras
        Hormiguero.setNObreras(Hormiguero.getNObreras()+1);

        //logger y timestamps
        Timestamp timestamp1 = new Timestamp(System.currentTimeMillis());
        TareaEscribir entrada1 = new TareaEscribir(Thread.currentThread().getName(), 0,
timestamp1);
        Escritor.logger.execute(entrada1);

        //entra por primera vez
        try {
            Hormiguero.entrar();
```

```
        }
        catch (InterruptedException ex)
        {
          interrumpido();
        }
        //las obreras pares
        if (id%2==0)
        {
           //funcionamiento de las obreras pares
           while (true)
           {
              //saca comida del almacen
              try
              {
                 Almacen.decStock(5, this);
              }
              catch (InterruptedException IE)
              {
                 interrumpido();
              }

              Timestamp timestamp = new Timestamp(System.currentTimeMillis());
              TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 6,
timestamp);
              Escritor.logger.execute(entrada);

              //camina del almacen al comedor
              Hormiguero.getMovimiento().add(this);
              try
              {
                 Thread.sleep((new Random().nextInt(3) + 1) * 1000);
              }
              catch(InterruptedException IE)
```

```
{
   interrumpido();
}
Hormiguero.getMovimiento().remove(this);

//deja la comida en el comedor
Hormiguero.getDejandoComida().add(this);
try
{
   Comedor.incStock(5);
}
catch (InterruptedException IE)
{
   interrumpido();
}
Hormiguero.getDejandoComida().remove(this);

//cada 10 iteraciones
if (iteracion%10==0&& iteracion !=0)
{
   //Entra a comer
   Hormiguero.getComer().add(this);
   try
   {
      Comedor.comer(1, 3);
   }
   catch(InterruptedException IE)
   {
      interrumpido();
   }
   Hormiguero.getComer().remove(this);
```

```
                //Entra a descansar
                Hormiguero.getDescanso().add(this);
                try
                {
                   Descanso.descansar(1);
                }
                catch(InterruptedException IE)
                {
                   interrumpido();
                }
                Hormiguero.getDescanso().remove(this);
             }
             //sube el contador de iteracion
             iteracion++;
          }
       }
       //las obreras impares
       else
       {
          //comportamiento de las obreras impares
          while (true)
          {
             Timestamp timestamp = new Timestamp(System.currentTimeMillis());
             TareaEscribir entrada = new TareaEscribir(Thread.currentThread().getName(), 3,
timestamp);
             Escritor.logger.execute(entrada);

             //sale del hormiguero
             try
             {
                Hormiguero.salir();
             }
```

```
catch (InterruptedException IE)
{
   interrumpido();
}

//recolecta comida
Hormiguero.getFuera().add(this);
try
{
   Thread.sleep(4000);
}
catch (InterruptedException IE)
{
   interrumpido();
}
Hormiguero.getFuera().remove(this);

//entra al hormiguero
try
{
   Hormiguero.entrar();
}
catch (InterruptedException IE)
{
   interrumpido();
}

//deja la comida en el almacen
try
{
   Almacen.incStock(5, this);
}
```

```
catch (InterruptedException IE)
{
   interrumpido();
}

//cada 10 iteraciones
if (iteracion%10==0&& iteracion !=0)
{
   //Entra a comer
   Hormiguero.getComer().add(this);
   try
   {
      Comedor.comer(1, 3);
   }
   catch(InterruptedException IE)
   {
      interrumpido();
   }
   Hormiguero.getComer().remove(this);

   //Entra a descansar
   Hormiguero.getDescanso().add(this);
   try
   {
      Descanso.descansar(1);
   }
   catch(InterruptedException IE)
   {
      interrumpido();
   }
   Hormiguero.getDescanso().remove(this);
}
```

```
            //aumenta las iteraciones del bucle
            iteracion++;
        }
    }
}

//comportamiento en caso de interrupcion
public void interrumpido()
{
    //solo se interrumpe por pausa
    try
    {
        latch.await();
    }
    catch (InterruptedException ex)
    {
        //por si acaso
        interrumpido();
    }
}
}
```

Soldado.java

```
package com.pablokarin.progav.part1.hilos;

import com.pablokarin.progav.log.Escritor;
import com.pablokarin.progav.log.TareaEscribir;
import com.pablokarin.progav.part1.*;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.CyclicBarrier;
```

```
public class Soldado implements Hormiga {
    private final int id;
    private int iteracion;
    private static CyclicBarrier barrera;
    private static CountDownLatch latch;
    private String nombre;
    private static final ArrayList<Soldado> listaSoldados = new ArrayList();

    public Soldado(int id)
    {
        //se añade a la lista de soldados estatica
        listaSoldados.add(this);

        //genera el nombre de la hormiga
        this.id = id;
              if (id < 10)
        {
            nombre = "HS000" + id;
        }
        else
        {
            if (id < 100)
            {
                nombre = "HS00" + id;
            }
            else
            {
                if (id<1000)
                {
                    nombre = "HS0" + id;
                }
                else
```

```
            {
                if (id<10000)
                {
                    nombre = "HS" + id;
                }
            }
        }
    }
}

@Override
public String getNombre()
{
    return nombre;
}

@Override
public void run()
{
    //pone nombre de la hormiga al hilo
    Thread.currentThread().setName(nombre);

    //actualiza las hormigas vivas
    Hormiguero.setNHormigasVivas(Hormiguero.getNHormigasVivas()+1);

    //logger y timestamps
    Timestamp timestamp1 = new Timestamp(System.currentTimeMillis());
    TareaEscribir entrada1 = new TareaEscribir(Thread.currentThread().getName(), 0,
timestamp1);
    Escritor.logger.execute(entrada1);

    //entra en el hormiguero por primera vez
    try
```

```
{
   Hormiguero.entrar();
}
catch (InterruptedException ex)
{
   interrumpido();
}

//bucle principal de comportamiento
while (true)
{
   //va a comer cada 6 iteraciones de comportamiento
   if (iteracion %6 == 0 && iteracion !=0)
   {
      //va a comer
      try
      {

         Hormiguero.getComer().add(this);
         Comedor.comer(1, 3);
         Hormiguero.getComer().remove(this);
      }
      catch(InterruptedException IE)
      {
         //gestion de listas
         boolean pausado = false;
         if (!Hormiguero.isPausa())
         {
            Hormiguero.getComer().remove(this);
         }
         else
         {
```

```
            pausado = true;
        }

        //interrupcion
        interrumpido();

        if (pausado) Hormiguero.getComer().remove(this);
    }
}
//comportamiento del resto de iteraciones
else
{
    //entrena
    try
    {
        Hormiguero.getInstruc().add(this);
        Instruc.instruir();
        Hormiguero.getInstruc().remove(this);
    }
    catch(InterruptedException IE)
    {
        //gestion de listas
        boolean pausado = false;
        if (!Hormiguero.isPausa())
        {
            Hormiguero.getInstruc().remove(this);
        }
        else
        {
            pausado = true;
        }
```

```
    //interrupcion
    interrumpido();

    if (pausado) Hormiguero.getInstruc().remove(this);
}

//descansa 2 segundos
try
{
    Hormiguero.getDescanso().add(this);
    Descanso.descansar(2);
    Hormiguero.getDescanso().remove(this);
}
catch(InterruptedException IE)
{
    //gestion de listas
    boolean pausado = false;
    if (!Hormiguero.isPausa())
    {
        Hormiguero.getDescanso().remove(this);
    }
    else
    {
        pausado = true;
    }

    //interrupcion
    interrumpido();

    if (pausado) Hormiguero.getDescanso().remove(this);

}
```

```
            }
            iteracion++;
        }
    }

    //gestor de interrupciones
    public void interrumpido()
    {
        //en caso de que sea pausa
        if(Hormiguero.isPausa())
        {
            try
            {
                //entra al countdownlatch de pausa
                latch.await();
            }
            catch (InterruptedException IE)
            {
                //por si acaso
                interrumpido();
            }
        }
        //en caso de que sea una amenaza
        else
        {
            Timestamp timestamp1 = new Timestamp(System.currentTimeMillis());
            TareaEscribir entrada1 = new TareaEscribir(Thread.currentThread().getName(), 10,
timestamp1);
            Escritor.logger.execute(entrada1);

            //sale a defender
            try
            {
```

```
    Hormiguero.salir();
}
catch (InterruptedException ex)
{
    //por si acaso
    interrumpido();
}
Hormiguero.getDefendiendo().add(this);

//entra en la cyclicbarrier de espera
try
{
    barrera.await();
}
catch(Exception e)
{
    //por si se pausa a mitad
    interrumpido();
}

//entra a la pelea (CountdownLatch)
try
{
    latch.await();
}
catch (InterruptedException IE)
{
    //por si se pausa a mitad
    interrumpido();
}

//vuelve a entrar en el hormiguero
```

```
        Hormiguero.getDefendiendo().remove(this);
        try{
            Hormiguero.entrar();
        }
        catch(InterruptedException e)
        {
          interrumpido();
        }
    }
}

//se llama desde el hormiguero cada vez que hay un ataque
public static void llamarAtaque(CyclicBarrier b, CountDownLatch l)
{
    barrera = b;
    latch = l;

    //interrumpe a todos los soldados
    for(int i = 0; i < listaSoldados.size(); i++)
    {
        String nombre = listaSoldados.get(i).getNombre();
        //busca en los hilos activos
        for (Thread t : Thread.getAllStackTraces().keySet())
        {
            //mira si tienen el mismo nombre y lo interrumpe
            if(t.getName().equals(nombre)) t.interrupt();
        }
    }
}

public static void setLatch(CountDownLatch latch) {
    Soldado.latch = latch;
```

```
    }



}
```

Cliente.java

```
package com.pablokarin.progav.pecl;


import com.pablokarin.progav.conexion.*;
import com.pablokarin.progav.jframe.*;
import java.rmi.*;


public class Cliente {


   public static void main(String[] args)
   {
      //crea la ventana
      VentanaRemota ventana = new VentanaRemota();
      ventana.setLocationRelativeTo(null);
      ventana.setVisible(true);
      try
      {
         InterfazOperaciones op = (InterfazOperaciones)
Naming.lookup("//127.0.0.1/ObjOperador");
         while(true)
         {
            //actualiza los contadores del cliente
            int nObrerasFuera = op.getNObrerasFuera();
            ventana.modificar(ventana.getCampoNObrerasFuera(),
Integer.toString(nObrerasFuera));
            int nObrerasDentro = op.getNObrerasDentro();
            ventana.modificar(ventana.getCampoNObrerasDentro(),
Integer.toString(nObrerasDentro));
            int nSoldadosInstruc = op.getNSoldadosInstruc();
```

```
            ventana.modificar(ventana.getCampoNSoldadosInstruc(),
Integer.toString(nSoldadosInstruc));

            int nSoldadosAmenaza = op.getNSoldadosAmenaza();

            ventana.modificar(ventana.getCampoNSoldadosAmenaza(),
Integer.toString(nSoldadosAmenaza));

            int nCriasComedor = op.getNCriasComedor();

            ventana.modificar(ventana.getCampoNCriasComedor(),
Integer.toString(nCriasComedor));

            int nCriasRefugiadas = op.getNCriasRefugio();

            ventana.modificar(ventana.getCampoNCriasRefugio(),
Integer.toString(nCriasRefugiadas));


         }
      }
      catch (Exception e)
      {
         System.out.println(e.getMessage());
      }


   }


   //genera una amenaza para la colonia
   //desde el cliente
   public static void amenaza()
   {
      try
      {
         InterfazOperaciones op = (InterfazOperaciones)
Naming.lookup("//127.0.0.1/ObjOperador");
         op.generarAmenaza();
      }
      catch(Exception e)
      {
         System.out.println(e.getMessage());
```

```
        }
    }
}
```

ScreenUpdate.java

```
package com.pablokarin.progav.pecl;

import com.pablokarin.progav.jframe.VentanaPrincipal;

//Este hilo se encarga de actualizar la pantalla por su cuenta

public class ScreenUpdate extends Thread
{
    private final VentanaPrincipal ventanaPrincipal;

    public ScreenUpdate(VentanaPrincipal ventana)
    {
        ventanaPrincipal = ventana;
    }

    @Override
    public void run()
    {
        this.setName("ScreenUpdater");
        while(true)
        {
            ventanaPrincipal.updateData();
        }
    }
}
```

Servidor.java

```
package com.pablokarin.progav.pecl;

import com.pablokarin.progav.conexion.Operador;
import com.pablokarin.progav.jframe.VentanaPrincipal;
import com.pablokarin.progav.part1.Hormiguero;
import com.pablokarin.progav.part1.hilos.*;
import java.io.FileWriter;
import java.io.IOException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Random;
import java.util.concurrent.locks.*;

public class Servidor {



   public static void main(String[] args)
   {
      //se inicia el servidor
      try
      {
         Operador op = new Operador();
         Registry reg = LocateRegistry.createRegistry(1099);
         Naming.rebind("//127.0.0.1/ObjOperador", op);
      }
      catch (Exception e)
      {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }
```

```
        //se muestra la pantalla
        VentanaPrincipal ventana = new VentanaPrincipal();
        ventana.setLocationRelativeTo(null);
        ventana.setVisible(true);

        //se crea el hilo que actualiza la pantalla
        ScreenUpdate screenUpdater = new ScreenUpdate(ventana);
        screenUpdater.start();

        //prepara el log
        String inicio ="=======COMIENZA LA EJECUCIÓN DEL PROGRMA=======";
        StringBuilder sb = new StringBuilder();
        sb.append((System.lineSeparator()));
        sb.append(inicio);
        sb.append((System.lineSeparator()));
        inicio = sb.toString();
        try
        {

            FileWriter fileWriter = new
FileWriter("src/main/java/com/pablokarin/progav/log/evolucionColonia.txt", true);
            //Se escribe en el log que comienza el programa
            fileWriter.write(inicio);
            fileWriter.close();
        }
        catch (IOException e) {System.out.println("No se pudo escribir en el archivo. Error de I/0:
" + e);}




        Lock l = new ReentrantLock();
        Condition c = l.newCondition();
```

```
//contadores
int soldados = 0;
int crias = 0;
int obreras = 0;

//generador de hormigas
for (int i = 1; i <= 10000; i++)
{
   //en caso de que esté pausado
   //el bucle no avanza
   if(Hormiguero.isPausa())
   {
      i--;
   }
   else
   {
      //espera para crear
      try
      {
         Thread.sleep(new Random().nextInt(2701)+800);
      }
      catch (InterruptedException IE)
      {
         System.out.println(IE.getMessage());
      }
      //comprueba si se ha parado la simulacion mientras descansaba
      if (!Hormiguero.isPausa())
      {
         if ((i%5)==0)
         {
```

```
                    //genera una soldado
                    new Thread(new Soldado(soldados)).start();
                    Hormiguero.aumentarSoldados();
                    soldados++;
                }
                else if (((i%5)-1)==0)
                {

                    //genera una cria
                    new Thread(new Cria(crias)).start();
                    crias++;
                }
                else
                {

                    //genera una obrera
                    new Thread(new Obrera(obreras)).start();
                    obreras++;
                }
            }
        }
    }
}
```