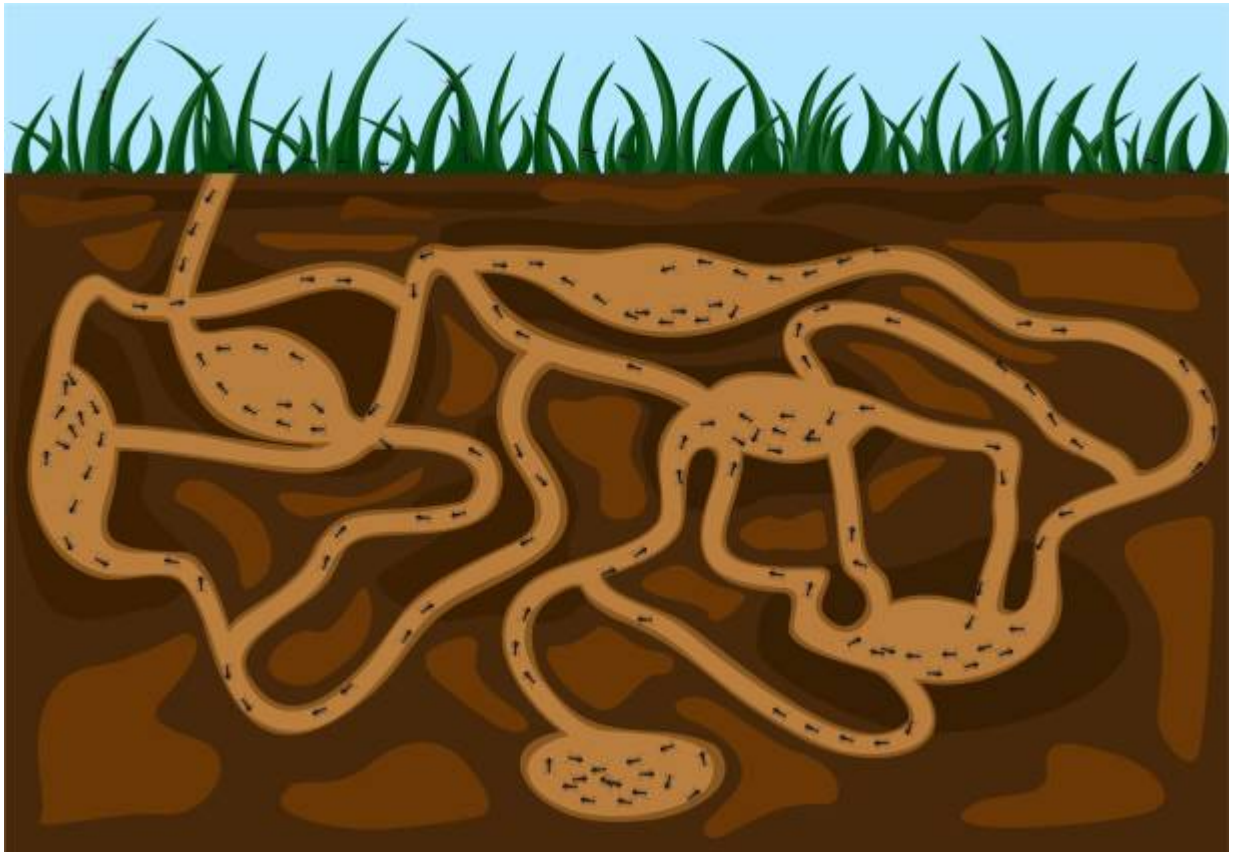

PROGRAMACION AVANZADA

DOCUMENTACIÓN PECL

Grado en Ingeniería Informática
Curso 2022/2023 - Convocatoria ordinaria
09125748S - Borrat Gutierrez, Roger
03202770C - Martín Calvo, Iván

Descripción del problema e identificación de los principales actores.

Dividiremos la explicación en dos partes, la primera parte irá en función de la parte 1 de la práctica y la segunda parte corresponderá a la segunda parte de la misma.

En la parte 1 se nos pide realizar un programa que simulará el comportamiento de unas hormigas en un hormiguero. Teniendo en cuenta que según su clase dentro del hormiguero actuarán de una manera u otra. Las hormigas pueden ser soldado, obrera o cría. Las crías únicamente se encargan de alimentarse y descansar principalmente, y en caso de amenaza se irá a refugiar. Las soldado basarán su comportamiento en alimentarse tras haber ido a la zona de instrucción y descansar 6 veces. Adicionalmente deben estar atentas para que en caso de amenaza dejasen lo que estuviesen haciendo e ir a defender el hormiguero, después volverán a lo que estaban haciendo. Por último las obreras que se encargaran de conseguir comida, llevarla al almacén en caso de tener un id impar y de llevar la comida del almacén a la zona de comer si es par, tras hacer 10 veces su rutina van a alimentarse y descansar. Cabe destacar que hay un único túnel de acceso al hormiguero con capacidad para 1 hormiga y otros 2 de salida, y que las hormigas se generarán en el exterior del hormiguero.

Todos estos sucesos deben verse reflejados en un archivo log.

En la parte 2 se tratará de una interfaz que actualiza sus datos en función de los de la interfaz de la parte 1, estas se verán afectadas mutuamente, ya que si la interfaz 2 detecta que se ha generado una amenaza, la primera lo tratará como si lo hubiese pulsado en ella. Los datos que se podrán ver será el número de hormigas de cierto tipo en una parte concreta del hormiguero.

Diseño general del sistema y discusión de herramientas de sincronización.

Vista la descripción anterior podemos diferenciar distintos tipos de clase en función de su función, las clases que representarán áreas como la zona de descanso, la zona de comer, la zona de instrucción, refugio… Las que representan las hormigas (hilos que definen en el método run su comportamiento) y las que ayudan al control o modificación de su comportamiento como puede ser la clase encargada de pausar la simulación de forma parcial, la que genera la amenaza o la que se encarga de realiza el archivo log. Adicionalmente para la parte dos se deberán de implementar las clases correspondientes para generar el comportamiento de RMI.

Como herramientas de sincronización emplearemos locks implicitos y explicitos (control de acceso a variables compartidas), conditions, monitores (esperas si son necesarias), semáforos (controlará que el número de hilos que accedan a dicha variable sea el deseado

dependiendo de su uso, además de que si fuese necesario mantendría una cola de cómo los hilos han ido llegando para que en el momento de acceso lo hagan en orden) y cyclic barriers(para la gestión de la amenaza).

El uso de estas herramientas fue distribuido dependiendo de su facilidad a la hora de implementarlo, como es el caso de los semáforos que facilitan el código necesario para hacer la entrada y salida. Otros como los bloqueos implícitos a la hora de definir acciones muy simples como métodos de incrementar, añadir a un arraylist… o explícitos a la hora de modificar la cantidad de alimentos en las zonas o monitores para causar esperas controladas en caso de falta de alimentos o si se ha pulsado el botón de pausa. El uso del cyclicbarrier es bastante claro a la hora de agrupar a las hormigas para defender una amenaza ya que se reinicia automáticamente cuando se cumple la condición pedida.

DESCRIPCIÓN DE LAS CLASES PRINCIPALES:

Clase Almacen:
Esta clase está compuesta por un int “total”, que almacenará la cantidad de comida en el almacén, un semáforo “entrar”, que se utilizará para que solo puedan entrar 10 hormigas obreras a la vez al almacén, un semáforo “em”, que garantizará la exclusión mutua,una ListaHormigas “lista”, que servirá para almacenar las hormigas obreras que se encuentren en el almacén,y un JTextField “j” para ir actualizando en la interfaz la cantidad de comida.

El método depositar, servirá para que una hormiga obrera deposite comida en el almacén. La hormiga entrará al almacén haciendo un aquire del semáforo entrar,añadirá su id a la lista de hormigas, dormirá de 2 a 4 segundos , hará un aquire de em para aumentar en 5 la variable total en exclusión mutua, actualizará el texto de j , hará un release de em, quitará su id de lista, y por último hará un release de entrar y llamará al método liberar que consiste en un notify para notificar a las hormigas que están esperando, por el método esperar, que contiene un monitor para garantizar que no se coja comida si no hay suficiente comida, que ya hay comida para que la cojan.

El método coger comida representa la acción de las hormigas para llevar la comida del almacén a la ZComer. Esto se consigue entrando al almacén igual que en depositar, con los dos acquire para garantizar la exclusión mutua y el máximo de hormigas dentro,se meterá el id de la hormiga en la lista, dormirá entre 1 y 2 segundos, se llama a la función esperar(explicada anteriormente) antes de hacer em.acquire , tras esto se disminuirá total en 5, se actualizará j, y se hará un release de los dos semáforos, para terminar quitando la hormiga de la lista.

Clase Amenaza:
La clase amenaza constará de dos arraylist de Soldados y Crias, para ir guardando las hormigas que más tarde se tendrán que interrumpir e ir al refugio o a repeler la amenaza, una ListaHormigas, que servirá para almacenar las hormigas que se encuentran repeliendo la amenaza, un booleano amenaza para indicar si hay una amenaza o no, un Cyclic Barrier c para que las soldado esperen a estar todas para salir a luchar, un JButton jb que será el botón en la interfaz que se utilizará para iniciar una amenaza, un Hormiguero h para que las soldado salgan y entren , un Refugio r para que las crias se refugien, un LogCreator lc para

almacenar en el documento log las diferentes acciones(decidimos ponerlo aquí haciendo una excepción, ya que normalmente se encuentra solo en las clases de las hormigas para facilitarnos el hecho de añadir al log en las excepciones), un JLabel txtEspera que indicará el estado de la pelea(Esperando o Peleando), y por último un int solRep para ir actualizando el número de soldados repeliendo la amenaza para poder enviaselo a cliente.

El método repeler representa cuando la soldado sale a repeler la amenaza. Lo primero que se hace es escribir en el log la acción de ir a repeler, tras esto, sale del hormiguero, se añade a la lista de soldados, incrementa el número de soldados repeliendo, espera a estar todas y tras esto se cambia el texto a peleando, pelan 20 segundos , se anula la interrupción, quitamos a la hormiga de la lista, decrementamos el número de soldados repeliendo, y si es la última hormiga en irse, volverá a poner el boton visible, indicará que ya no hay amenaza y se notificará a las crias, se indicará el fin el log y txtEspera se “borrará”, volverá a entrar al hormiguero, y se escribirá en el log que ha repelido la amenaza.

El método refugiarse, representa cuando las crías se van al refugio por una amenaza.Primero la hormiga se va al refugio, indicandolo en el log, tras esto esperará mientras haya una amenaza con el metodo dormir que es un monitor, cuando la última soldado haya terminado les hará un notifyAll, tras lo que la cría saldrá del refugio, indicandolo en el log, y finalizando su interrupción.

El método amenaza, sirve para interrumpir las hormigas soldado y cría cuando se pulsa el botón de la interfaz, indicando que hay una amenaza.Lo primero que se hace es indicar en el log que hay una amenaza, tras esto se pone amenaza a true para indicar que hay una amenaza, el botón se vuelve invisible para que no pueda haber otra amenaza hasta que termine, tras esto crearemos el Cyclic barrier con el tamaño del número de soldados que haya en el momento, e interrumpiremos a las soldado y cría que había hasta el momento de la amenaza para que hagan sus respectivas tareas en caso de amenaza.

Clase Detener:
Esta clase está compuesta por un booleano que indicará si se ha detenido iteración.

Tendrá un método detener, que pondrá el booleano a true, un método esperar, que constará de un monitor que hará que las hormigas esperen mientras que esté detenido y un método reanudar que pondrá detener a false y notificará a las hormigas que pueden seguir con su iteración.

Clase Cliente:
La clase cliente constará de 6 JTextField, uno para cada dato que se quiere mostrar, un booleano amenaza, que indicará si hay una amenaza, y un JButton botAmenaza, que será el boton encargado de crear la amenaza remotamente.Esta clase será un hilo que actualizará los valores de sus textfields constantemente.

El método run, creará constantemente un obj InterfaceColonia y se conectará al servidor, para actualizar sus textfiel con los datos que le envie el servidor mediante los metodos de la interfaz.Si ha habido una amenaza desde el servidor, el botón del cliente se volverá invisible hasta que esta termine. Por otro lado, si se ha pulsado el botón en cliente(haciendo que amenaza esté a true), el botón se pondrá a true y se llamará a la función causarAmenaza

de la clase InterfaceColonia, que mas tarde detallaremos.Si no ha ocurrido ninguna de las dos acciones anteriores, simplemente pondremos el botón visible. Por último dormiremos el hilo 0,1 segundo para que haya un pequeño retraso y observar mejor los datos.

Clase Colonia:
La clase Colonia, servirá para desarrollar los métodos que se usarán en la comunicación entre servidor y cliente.Esta clase estará compuesta por una Amenaza, un Hormiguero, un Refugio, una ZComer y una ZInstruccion.Extenderá UnicastRemoteObject e implementará InterfaceColonia para que se puedan llamar a sus métodos remotamente mediante la interface InterfaceColonia

Sus métodos, que lanzarán la excepción RemoteException para poder ser llamados remotamente, servirán para obtener el número de hormigas obras fuera y dentro, el número de soldados haciendo instrucción o repeliendo una invasión, el número de crías en ZComer o en el refugio, obtener si hay una amenaza y por último un método para provocar una amenaza que llamará a la función amenaza().

Clase CreadorHormigas:
La clase CreadorHormigas servirá para crear un hilo que vaya creando las hormigas de forma ordenada, para que las hormigas puedan ir haciendo sus tareas mientras se siguen creando las demás. Esta clase constará de todos los elementos necesarios para crear a las diferentes hormigas(Almacen, Hormiguero,Amenaza…).

El método run servirá para ir creando las hormigas. Tendrá unos contadores de las hormigas para añadirlo al id de cada hormiga, tendrá un bucle for que llegará a 2000, dentro tendrá un bucle for de 3 para crear una soldado y una cría cada 3 obreras, cada vez que se cree una hormiga se incrementará su respectivo contador y se mirará si la iteración está detenida, en cuyo caso esperará a que se reanude, por último se dormirá entre 0,8 y 3,5 segundos para que la creación sea escalonada.

Clase Hormiga_Cria:
La clase Hormiga_Cria, representará la actuación de una cría, mediante un hilo. Esta clase tendrá como atributos, todas las zonas que sean relevantes para la iteración de una hormiga cría(Hormiguero, ZDescanso…), además tendrá un string id, para identificar a la hormiga,también tendrá un LogCreator para ir almacenando en el log las acciones de la hormiga.

El método run, lo primero que hará será almacenar en el array de crias en amenaza a la cria que se haya creado, para después, si hay una amenaza interrumpirla, tras esto se indicará en el log que se ha creado, entrará en el hormiguero, indicandolo en el log, y mirará si hay una amenaza, en cuyo caso se refugiará directamente,si no mirará si se ha detenido la iteración(cosa que hará siempre entre funciones),y empezará su rutina habitual, cojera comida, indicandolo en el log, comerá, tardando entre 3 y 5 segundos, indicandolo en el log y por último descansará 4 segundos

Clase Hormiga_Obrera:
La clase Hormiga_Obrera, representará la actuación de una soldado, mediante un hilo. Esta clase tendrá como atributos, todas las zonas que sean relevantes para la iteración de una

hormiga obrera(Hormiguero, ZDescanso…), además tendrá un string id, para identificar a la hormiga,también tendrá un LogCreator para ir almacenando en el log las acciones de la hormiga y un int n para ver si la hormiga es par o impar y dependiendo de eso hará una cosa u otra.

El método run empezará creando un contador a cero para ir contando el número de iteraciones realizadas por la hormiga, tras eso se indicará que se ha creado la hormiga en el log(acción que ocurrirá siempre después de que la hormiga haya hecho alguna de sus tareas) , y se aumentará el contador de obreras fuera de la colonia,(ya que se crean fuera), tras eso entrará a la colonia disminuyendo el contador de hormigas fuera y aumentando el de dentro y mirará si está pausado esperando si esto fuera así, hasta que se reanude(esto lo hará después de cada tarea), tras esto empezará su rutina. En caso de haber hecho 10 iteraciones, cogerá comida, la comerá tardando 3 segundos y descansará 1 segundo, y reiniciará su contador. Si no lleva 10 iteraciones, si el número de su id(n) es impar, saldrá del hormiguero, decrementando el número de obreras dentro e incrementando el número de obreras fuera, cogerá comida, entrará al hormiguero disminuyendo el contador de hormigas fuera y aumentando el de dentro y depositará la comida en el almacén, por otro lado, si su número es par, cogerá comida del almacén, irá a ZComer y depositará allí la comida.Por último, aumentará su contador.

Clase Hormiga_Soldado:
La clase Hormiga_Soldado, representará la actuación de una soldado, mediante un hilo. Esta clase tendrá como atributos, todas las zonas que sean relevantes para la iteración de una hormiga soldado(Hormiguero, ZDescanso…), además tendrá un string id, para identificar a la hormiga,también tendrá un LogCreator para ir almacenando en el log las acciones de la hormiga.

El método run comenzará iniciando un contador de iteraciones a 0, tras esto se indicará en el log que la hormiga a entrado(esto ocurrirá tras cada tarea de la hormiga), la soldado entrará al hormiguero, y mirará si la iteración se ha detenido en cuyo caso se pausará hasta que se reanude la iteración(esto ,lo hará tras cada tarea), tras esto empezará la rutina de la soldado.Primero mirará si ya ha hecho 6 iteraciones, en cuyo caso cogerá comida de ZComer y comerá tardando 3 segundos, por último reiniciará su contador, si no, hará instrucción, descansará 2 segundos y aumentará su contador.

Clase Hormiguero:
La clase hormiguero consta de dos semáforos uno iniciado a 1, para representar el unico tunel de entrada, y otro iniciado a 2 para representar los dos túneles de salida, ambos iniciados a true para que se respete el orden, además tendrá dos ListaHormigas, una para guardar las hormigas que están recogiendo comida, y otra para representar,las que están yendo a dejar comida a ZComer, también le entrará amenaza para que en el caso de interrupción, se pueda llamar a sus funciones, y también tendrá dos contadores uno para contar las obreras fuera de la colonia y otra para las que están dentro.
La función entrar, representa la acción de entrar al hormiguero, haciendo un acquire de entrada, tardando 0.1 segundos en entrar,para terminar haciendo un release de entrada. En el catch se contempla la posibilidad de que haya habido una amenaza y el hilo que estaba entrando se haya interrumpido, en cuyo caso, se mirará que el segundo caracter del string que le entra a la función(que será el id de la hormiga que estaba entrando), en caso de ser

una S, será una soldado y se llamará a repeler, si no será una cría y se llamará a la función refugiar(**este catch, será igual para todas las funciones que sean mutuas para cria y soldado y que estén relacionadas con la iteración de estas**).

El método cogerComida representa la recolección de comida por parte de las obreras fuera de la colonia. Lo primero que se hará será añadir a lista, el id de la obrera que está recogiendo. La recoleccion tardará 3.9 segundos(+ los 0.1 del tunel). Por último, se eliminará a la obrera de lista.

El método salir, representa cuando una hormiga sale, esto solo lo harán las obreras. Lo primero que harán será un acquire de salida “bloqueando uno de los túneles”, tardarán 0.1 segundos en entrar y harán un release “desbloqueando ese túnel”.

El método irZComer, representa el trayecto que hacen las obreras para representar las hormigas que están transportando comida del almacén a la ZComer. Primero se añadirá a la lista caminoZcomer el id de la obrera que está haciendo esta tarea.

Interface InterfaceColonia:
La interface InterfaceColonia, nos servirá para la comunicación entre el servidor y el cliente, guardando los metodos remotos a los que cliente llamará remotamente explicados en la clase colonia.

Clase ListaHormigas
La clase ListaHormigas servirá para crear listas de hormigas, y poder rellenar los diferentes textfields de la interfaz con los ids de hormigas correspondientes. Tendrá como atributos un ArrayList de strings para almacenar los id, y un JTextField para actualizar con su lista.

El método añadir, añadirá al ArrayList, el id indicado en la entrada, y actualizará el textfield añadiendo este id. Este método es synchronized para garantizar la exclusión mutua para el array y el textfield.

El método quitar, quitará del ArrayList, el id indicado y actualizará el texto del textfield al nuevo contenido de la lista , imprimiendo con un for todos los id de la lista. Este método es synchronized para garantizar la exclusión mutua para el array y el textfield.

Clase LogCreator:

La clase LogCreator, nos servirá para poder escribir en un documento txt, lo que está sucediendo en nuestro programa. Esta clase constará de un FileWriter(que llamaremos evolucionColonia.txt)para poder escribir en este y de un Lock para garantizar que la exclusión mutua del FileWritter

El método añadir_aLog, comenzará haciendo un lock, para hacer la escritura en exclusión mutua, tras esto se guardará la fecha en una variable local y se indicará el FileWriter en el que se va a escribir poniendo este a true, tras esto haremos un write, metiendo como valores la fecha y el string que le entra como parámetro, con un espacio entre ellos y con un salto de línea al final. Después cerraremos el FileWriter y finalmente hará el unlock.

Clase Refugio:

La clase refugio representa el refugio al que irán las crías en caso de amenaza. Estará compuesta por una ListaHormigas para almacenar las crías que se encuentran en el refugio y poder actualizar el textfield de esta lista, y un int que contará el numero de hormigas en el refugio.
Tendrá métodos synchronized para incrementar y decrementar el contador en exclusión mutua, tal y como también los tienen las clases en las que hay contadores para después enviarlos al cliente.

El método irRefugio, representa cuando una cría se mete al refugio, incrementando el contador y añadiendo el id de esta hormiga a la lista.

El método salirRefugio, representa cuando la cría sale del refugio porque ya se ha repelido la amenaza, eliminando el id de la cría de la lista y decrementando el contador.

Clase Servidor:
Este hilo es el encargado de mantener una conexión con Cliente con el fin de mantener un sistema distribuido, permitiendo al cliente actualizar sus datos e interactuar con la interfaz_parte_1. Esta clase está compuesta por los objetos Amenaza, Hormiguero, Refugio, ZComer y ZInstruccion.

El método run() crea un objeto Colonia al que se le pasan los atributos de Servidor, para que pueda acceder a sus datos. Después hace un registry con el fin de buscar objetos que están siendo exportados para su uso y finalmente espera a que un cliente se conecte con el Naming.rebind al que se le pasa el nombre del registro remoto y el objeto asociado a este.

Clase ZComer:
Esta clase está compuesta por el contador comida que determina la cantidad de alimentos que se encuentran en la zona, un lock para garantizar la exclusión mutua, un condition para tratar las hormigas en caso de que no haya comida, un JTextField para mostrar en la interfaz la cantidad de alimentos que hay, un atributo ListaHormigas el cual se inicializa en el constructor pasándole un JTextField para que a la hora de añadir una hormiga que vaya acceder a la zona, se vea reflejado en la interfaz, un objeto amenaza para tratar el caso de capturar la excepción en caso de que se genere una amenaza y un contador criaZcomer para determinar el número de hormigas en la zona.

Los métodos incrementarCriaZC() y decrementarCriaZC() se encargan de sumar y restar 1 a criaZcomer respectivamente de forma concurrente para que sea segura su variación al ser synchronized.

El método depositar realiza un bloqueo para depositar la comida en la zona, lo cual tarda entre 1 y 2s, añadiendo 5 alimentos y despierta a las hormigas que estuvieran esperando por la comida. Finalmente quita el bloqueo.

El método cogerComida(String id) el cual actualiza las hormigas que están en la zona y se encarga de dormir a las hormigas en el caso de que no haya comida para estas esperando a que alguna otra deposite comida para despertarla y que pueda modifica la cantidad de comida que está en la zona. También trata el caso de amenaza como otros de los métodos anteriores.

El método comer que simplemente es un sleep de la cantidad que necesite cada una de las hormigas para comer. También trata el caso de amenaza como otros de los métodos anteriores.

El método getCriaZcomer es un getter del parámetro criaZomer.

Clase ZDescanso:
Compuesta por un atributo ListaHormigas el cual se inicializa en el constructor pasándole un JTextField para que a la hora de añadir una hormiga que vaya a descansar, se vea reflejado en la interfaz y un objeto amenaza para tratar el caso de capturar la excepción en caso de que se genere una amenaza

El método descansar(int n, String id) principalmente hace un sleep de n segundos dependiendo del parámetro que se le pase a la hora de llamarlo. Posee un catch para que en el caso de que una cría o una soldado estén descansando, en el momento de que haya una amenaza realicen las acciones necesarias.

Clase ZInstruccion:
Esta clase está compuesta por un atributo ListaHormigas el cual se inicializa en el constructor pasándole un JTextField para que a la hora de añadir una soldado que este haciendo la instrucción se vea reflejado en la interfaz, un objeto amenaza para tratar el caso de capturar la excepción en caso de que se genere una amenaza y el contador para saber el número de soldados en esta zona.

Los métodos synchronized incrementarSoII() y decrementarSoII() se encargan de sumar y restar 1 respectivamente de forma concurrente para que sea segura su variación.

El método hacer_Instruccion(String id) se encarga de modificar el contador y la lista al principio y al final del proceso, el proceso principal se ve determinado por un sleep de 2 a 8s. Además trata la excepción en caso de que se genere una amenaza, decrementando el contador de soldados haciendo instrucción, quitando el id de la lista de soldados haciendo instruccion, llamando a la función repeler, en cuanto termine de repeler, volverá a su trabajo por lo que por último en el catch, volveremos a añadir el id a la lista.

Clase Interfaz_parte_1:
Esta clase se ve compuesta principalmente por el conjunto de JTextFields que se emplean para mostrar la posición en cada momento de cada hormiga y los JButton que tienen tanto la funcionalidad de causar una amenaza como de pausar y reanudar la simulación. También hay variables empleadas a la hora de gestionar el inicio y funcionamiento de la simulación como son las clases ClaseDetener, Almacen, Hormigueo, Refugio, ZDescanso, ZInstruccion, ZComer, Amenaza, LogCreator y Servidor. Todas las anteriores menos Servidor se ven inicializadas con lo necesario según se explica en apartados anteriores en

el método initComponents() y son pasadas como argumentos al hilo CreadorHormigas que se encargará de empezar la simulación. Además se le pasan Amenaza, Hormiguero, Refugio, ZComer y ZInstruccion a Servidor para quien se encargará de mantener una conexión con la Interfaz_parte_2 y enviarle los datos necesarios de cada una de estas zonas.

Clase Interfaz_parte_2:
Esta clase únicamente posee los JTextField que se le pasan al hilo cliente junto al JButton con el fin de que este hilo al ejecutarse sea capaz de mostrar los datos de la otra interfaz y sea capaz de interactuar con la misma para poder causar una amenaza.

Diagrama de clases

Paquete Logica

Se adjunta con la documentación dado su gran tamaño

Paquete Interfaz

Se adjunta con la documentación dado su gran tamaño

Código del proyecto

Clase Almacen

```
import java.util.concurrent.Semaphore;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;


public class Almacen {
    private int total;
    private Semaphore entrar;
    private Semaphore em;
    private ListaHormigas lista;
    private JTextField j;

    public Almacen(JTextField j1, JTextField j2) {
        total = 0;
        entrar = new Semaphore(10, true);
        em = new Semaphore(1);
        this.j = j2;
        j.setText(String.valueOf(total));
        lista = new ListaHormigas(j1);
    }
```

```
    public void depositar(String id){
        try{
            entrar.acquire();
             lista.añadir(id);
            Thread.sleep(2000 + (int)(Math.random()*2000));
            em.acquire();
            total+=5;
            j.setText(String.valueOf(total));
            em.release();
            lista.quitar(id);
            entrar.release();
            liberar();
        }catch(InterruptedException e){System.out.println("ERROR")

    }

    public void cogerComida(String id){
        try {
            entrar.acquire();
            lista.añadir(id);
            Thread.sleep(1000 + (int)(Math.random()*1000));
            esperar();
            em.acquire();
            total-=5;
            j.setText(String.valueOf(total));
            em.release();
            entrar.release();
            lista.quitar(id);
        } catch (InterruptedException ex) {
            Logger.getLogger(Almacen.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    public synchronized void esperar(){
        while(total<5){
            try {
                wait();
            } catch (InterruptedException ex) {
                Logger.getLogger(Almacen.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    public synchronized void liberar(){
        notify();
    }
}
```

Clase Amenaza

```
import java.util.ArrayList;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Amenaza {
    private ArrayList<Hormiga_Soldado> hs = new ArrayList<>();
    private ArrayList<Hormiga_Cria> hc = new ArrayList<>();
    private ListaHormigas soldados;
    private Boolean amenaza = false;
    private CyclicBarrier c;
    private JButton jb;
    private Hormiguero h;
    private Refugio r;
    private LogCreator lc;
    private JLabel txtEspera;
    private int solRep = 0;

    public synchronized void incrementarSolR(){
        solRep++;
    }

    public synchronized void decrementarSolR(){
        solRep--;
    }


    public Amenaza(JButton jb, JTextField jtf, Hormiguero h, Refugio r, LogCreator lc, JLabel
txtEsp){
        this.jb = jb;
        soldados =  new ListaHormigas(jtf);
        this.h = h;
        this.r = r;
        this.lc = lc;
        this.txtEspera = txtEsp;
    }

    public synchronized void añadir_soldado(Hormiga_Soldado h){
        hs.add(h);
    }

    public synchronized void añadir_cria(Hormiga_Cria h){
```

```
    hc.add(h);
}

public void repeler(String id){
    try {
        lc.añadir_aLog(id + " va a repeler la amenaza");
        h.salir();
        soldados.añadir(id);
        incrementarSolR();
        c.await();
        txtEspera.setText("Peleando");
        Thread.sleep(20*1000);
        Thread.currentThread().interrupted();
        soldados.quitar(id);
        decrementarSolR();
        if(soldados.getLista().isEmpty()){
            jb.setVisible(true);
            amenaza = false;
            despertarCrias();
            lc.añadir_aLog("SE HA REPELIDO LA AMENAZA!!! :)");
            txtEspera.setText("   ");
        }
        h.entrar(id);
        lc.añadir_aLog(id + " ha repelido la amenaza");
    } catch (InterruptedException ex) {
        Logger.getLogger(Amenaza.class.getName()).log(Level.SEVERE, null, ex);
    } catch (BrokenBarrierException ex) {
        Logger.getLogger(Amenaza.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public Boolean getAmenaza() {
    return amenaza;
}

public synchronized void despertarCrias(){
    notifyAll();
}

public synchronized void dormir(){
    while(amenaza){
        try {
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Amenaza.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```
    public void refugiar(String id){
        r.irRefugio(id);
        lc.añadir_aLog(id + " ha llegado al refugio por la amenaza");
        dormir();
        r.salirRefugio(id);
        lc.añadir_aLog(id + " ha salido del refugio");
        Thread.currentThread().interrupted();
    }

    public void amenaza(){
        try{
            lc.añadir_aLog("HAY UNA AMENAZA!!! :(");
            amenaza = true;
            jb.setVisible(false);
            int tam = hs.size();
            c = new CyclicBarrier(tam);
            for(int i = 0; i < tam ; i++){
                hs.get(i).interrupt();

            }
            txtEspera.setText("Esperando Soldados");
            tam = hc.size();
            for(int i = 0; i<tam ; i++){
                hc.get(i).interrupt();
            }
        }catch(Exception e){
            jb.setVisible(true);
        }
    }

    public int getSolRep() {
        return solRep;
    }
}
```

Clase ClaseDetener

```
import java.util.logging.Level;
import java.util.logging.Logger;

public class ClaseDetener {
    private Boolean detener = false;

    public synchronized void esperar(){

        while(detener){
            try {
                wait();
```

```
            } catch (InterruptedException ex) {
                Logger.getLogger(ClaseDetener.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    public void detener(){
        detener = true;
    }

    public synchronized void reanudar(){
        detener = false;
        notifyAll();
    }
}
```

Clase Cliente

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JTextField;

public class Cliente extends Thread {
    private JTextField extCol;
    private JTextField intCol;
    private JTextField hacInst;
    private JTextField repeliendo;
    private JTextField criaCom;
    private JTextField criaRef;
    private boolean amenaza = false;
    private JButton botAmenaza;

    public Cliente(JTextField extCol, JTextField intCol, JTextField hacInst, JTextField
repeliendo, JTextField criaCom, JTextField criaRef, JButton botAmenaza) {
        this.extCol = extCol;
        this.intCol = intCol;
        this.hacInst = hacInst;
        this.repeliendo = repeliendo;
        this.criaCom = criaCom;
        this.criaRef = criaRef;
        extCol.setText("0");
        intCol.setText("0");
        hacInst.setText("0");
        repeliendo.setText("0");
```

```
        criaCom.setText("0");
        criaRef.setText("0");
        this.botAmenaza = botAmenaza;
    }

    @Override
    public void run(){
      while(true)
      {
          try
          {
            InterfaceColonia obj = (InterfaceColonia)
Naming.lookup("//127.0.0.1/ObjetoColonia");
            extCol.setText(String.valueOf(obj.obExterior()));
            intCol.setText(String.valueOf(obj.obInterior()));
            hacInst.setText(String.valueOf(obj.solInstruccion()));
            repeliendo.setText(String.valueOf(obj.solInvasion()));
            criaCom.setText(String.valueOf(obj.criZonaCom()));
            criaRef.setText(String.valueOf(obj.criRefugio()));
            if(obj.pasarAmenaza()){
               botAmenaza.setVisible(false);
            }else if(amenaza){
               botAmenaza.setVisible(false);
               obj.causarAmenaza();
               amenaza = false;
            }else{
               botAmenaza.setVisible(true);
            }
            Thread.sleep(100);
          } catch (NotBoundException | MalformedURLException | RemoteException |
InterruptedException ex) {
              Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
          }
      }
    }

    public void setAmenaza(boolean amenaza) {
        this.amenaza = amenaza;
    }
}
```

Clase Colonia

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;



public class Colonia extends UnicastRemoteObject implements InterfaceColonia{
```

```
    private Amenaza am;
    private Hormiguero h;
    private Refugio r;
    private ZComer zc;
    private ZInstruccion zi;

    public Colonia(Amenaza am, Hormiguero h, Refugio r, ZComer zc, ZInstruccion zi) throws
RemoteException {
        this.am = am;
        this.h = h;
        this.r = r;
        this.zc = zc;
        this.zi = zi;
    }

    @Override
    public int obExterior() throws RemoteException{
        return h.getObfuera();
    }

    @Override
    public int obInterior() throws RemoteException{
        return h.getObdentro();
    }

    @Override
    public int solInstruccion() throws RemoteException{
        return zi.getSolInst();
    }

    @Override
    public int solInvasion() throws RemoteException{
        return am.getSolRep();
    }

    @Override
    public int criZonaCom() throws RemoteException{
        return zc.getCriaZcomer();
    }

    @Override
    public int criRefugio() throws RemoteException{
        return r.getCriaRef();
    }

    @Override
    public void causarAmenaza() throws RemoteException{
        am.amenaza();
```

```
    }

    @Override
    public boolean pasarAmenaza() throws RemoteException{
        return am.getAmenaza();
    }
}
```

Clase CreadorHormigas

```
public class CreadorHormigas extends Thread{
    private Almacen a ;
    private Hormiguero h;
    private Refugio r;
    private ZComer c;
    private ZDescanso d;
    private ZInstruccion i;
    private ClaseDetener cd;
    private Amenaza am;
    private LogCreator lc;




    public CreadorHormigas(Almacen a, Hormiguero h, Refugio r, ZComer c, ZDescanso d,
ZInstruccion i, ClaseDetener cd, Amenaza am, LogCreator lc) {
        this.a = a;
        this.h = h;
        this.r = r;
        this.c = c;
        this.d = d;
        this.i = i;
        this.cd = cd;
        this.am = am;
        this.lc = lc;
    }

    public String num_ceros(String id){
        int len = id.length();
        int ceros = 4-len;
        String res ="";
        for(int i = 0; i<ceros;i++){
            res+="0";
        }
        res+=id;
        return res;
    }
```

```
    public void run(){

        int cont_o = 1;
        int cont_s = 1;
        int cont_c = 1;
        try{
            for(int j = 0; j<2000 ; j++){
                for(int k = 0; k<3 ;k++){
                    Hormiga_Obrera o = new Hormiga_Obrera("HO" +
num_ceros(String.valueOf(cont_o)), h, d, c, a, cont_o, cd, lc);
                    o.start();
                    cont_o++;
                    cd.esperar();
                    Thread.sleep(800 + (int)(Math.random()*2700));
                }
                Hormiga_Soldado s = new Hormiga_Soldado("HS" +
num_ceros(String.valueOf(cont_s)), h, d, c, i, cd, am, lc);
                s.start();
                cont_s++;
                cd.esperar();
                Thread.sleep(800 + (int)(Math.random()*2700));
                Hormiga_Cria c1 = new Hormiga_Cria("HC" + num_ceros(String.valueOf(cont_c)),
h, d, c, cd, am, lc);
                c1.start();
                cont_c++;
                cd.esperar();
                Thread.sleep(800 + (int)(Math.random()*2700));
            }
        }catch(InterruptedException e){System.out.println(e);}

    }
}
```

Clase Hormiga_Cria

```
public class Hormiga_Cria extends Thread{
    private String id;
    private Hormiguero hormiguero;
    private ZDescanso zdescanso;
    private ZComer zcomer;
    private ClaseDetener cd;
    private Amenaza a;
    private LogCreator lc;

    public Hormiga_Cria(String id, Hormiguero h, ZDescanso z1, ZComer z2, ClaseDetener
cd, Amenaza a, LogCreator lc) {
        this.id = id;
        this.hormiguero = h;
```

```
        this.zdescanso = z1;
        this.zcomer = z2;
        this.cd = cd;
        this.a = a;
        this.lc = lc;
    }

    @Override
    public void run(){
        a.añadir_cria(this);
        lc.añadir_aLog("Se ha generado la hormiga: "+id);
        hormiguero.entrar(id);
        lc.añadir_aLog(id + " ha entrado al hormiguero");
        if(a.getAmenaza()){
            a.refugiar(id);
        }
        cd.esperar();
        while(true){
            zcomer.cogerComida(id);
            lc.añadir_aLog(id + " coge comida");
            cd.esperar();
            zcomer.comer(3000+(int)(Math.random()*2000), id);
            lc.añadir_aLog(id + " ha comido");
            cd.esperar();
            zdescanso.descansar(4000, id);
            lc.añadir_aLog(id + " ha descansado");
            cd.esperar();
        }

    }
}
```

Clase Hormiga_Obrera

```
public class Hormiga_Obrera extends Thread{
    private Almacen almacen;
    private String id;
    private Hormiguero hormiguero;
    private ZDescanso zdescanso;
    private ZComer zcomer;
    private int n;
    private ClaseDetener cd;
    private LogCreator lc;

    public Hormiga_Obrera(String id, Hormiguero h, ZDescanso z1, ZComer z2, Almacen a,
int n, ClaseDetener cd, LogCreator lc) {
        this.id = id;
        this.hormiguero = h;
        this.zdescanso = z1;
```

```
    this.zcomer = z2;
    this.almacen = a;
    this.n = n;
    this.cd = cd;
    this.lc = lc;
}

@Override
public void run(){
    int cont = 0;
    lc.añadir_aLog("Se ha generado la hormiga: "+id);
    hormiguero.incrementarObF();
    hormiguero.entrar(id);
    hormiguero.decrementarObF();
    hormiguero.incrementarObD();
    lc.añadir_aLog(id + " ha entrado al hormiguero");
    cd.esperar();
    while (true){
        if(cont==10){
            zcomer.cogerComida(id);
            lc.añadir_aLog(id + " ha cogido comida");
            cd.esperar();
            zcomer.comer(3000, id);
            lc.añadir_aLog(id + " ha comido");
            cd.esperar();
            zdescanso.descansar(1000, id);
            lc.añadir_aLog(id + " ha descansado");
            cont=0;
            cd.esperar();
        }else{
            if(n%2==1){
                hormiguero.salir();
                hormiguero.decrementarObD();
                hormiguero.incrementarObF();
                lc.añadir_aLog(id + " ha salido del hormiguero");
                cd.esperar();
                hormiguero.cogerComida(id);
                lc.añadir_aLog(id + " ha recogido comida del exterior");
                cd.esperar();
                hormiguero.entrar(id);
                hormiguero.decrementarObF();
                hormiguero.incrementarObD();
                lc.añadir_aLog(id + " ha entrado al hormiguero");
                cd.esperar();
                almacen.depositar(id);
                lc.añadir_aLog(id + " ha guardado la comida en el almacen");
                cd.esperar();
            }else{
```

```
                almacen.cogerComida(id);
                lc.añadir_aLog(id + " ha cogido comida del almacen");
                cd.esperar();
                hormiguero.irZComer(id);
                lc.añadir_aLog(id + " ha ido a la zona para comer");
                cd.esperar();
                zcomer.depositar(id);
                lc.añadir_aLog(id + " ha dejado la comida");
                cd.esperar();
            }cont++;
        }
    }
  }
}
```

Clase Hormiga_Soldado

```
public class Hormiga_Soldado extends Thread{
   private ZInstruccion zinst;
   private String id;
   private Hormiguero hormiguero;
   private ZDescanso zdescanso;
   private ZComer zcomer;
   private ClaseDetener cd;
   private Amenaza a;
   private LogCreator lc;

   public Hormiga_Soldado(String id, Hormiguero h, ZDescanso z1, ZComer z2,
ZInstruccion z, ClaseDetener cd, Amenaza a, LogCreator lc) {
      this.id = id;
      this.hormiguero = h;
      this.zdescanso = z1;
      this.zcomer = z2;
      this.zinst = z;
      this.cd = cd;
      this.a = a;
      this.lc = lc;
   }

   @Override

   public void run(){
      int cont = 0;
      lc.añadir_aLog("Se ha generado la hormiga: "+id);
      a.añadir_soldado(this);
      hormiguero.entrar(id);
      lc.añadir_aLog(id + " ha entrado al hormiguero");
      cd.esperar();
      while(true){
```

```
            if(cont == 6){
                zcomer.cogerComida(id);
                lc.añadir_aLog(id + " ha cogido comida de la zona para comer");
                cd.esperar();
                zcomer.comer(3000, id);
                lc.añadir_aLog(id + " ha comido");
                cd.esperar();
                cont = 0;
            }else{
                zinst.hacer_Instruccion(id);
                lc.añadir_aLog(id + " ha hecho la instruccion");
                cd.esperar();
                zdescanso.descansar(2000, id);
                lc.añadir_aLog(id + " ha descansado");
                cont++;
                cd.esperar();
            }
        }
    }
}
```

Clase Hormiguero

```
import java.util.concurrent.Semaphore;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;

public class Hormiguero {
    private Semaphore entrada;
    private Semaphore salida;
    private ListaHormigas lista;
    private ListaHormigas caminoZcomer;
    private Amenaza amenaza;
    private int obfuera = 0;
    private int obdentro = 0;

    public Hormiguero(JTextField j1, JTextField j2, Amenaza am) {
        entrada = new Semaphore(1, true);
        salida = new Semaphore(2, true);
        lista = new ListaHormigas(j1);
        caminoZcomer = new ListaHormigas(j2);
        this.amenaza = am;
    }

    public synchronized void incrementarObF(){
        obfuera++;
    }
```

```
public synchronized void incrementarObD(){
   obdentro++;
}

public synchronized void decrementarObF(){
   obfuera--;
}

public synchronized void decrementarObD(){
   obdentro--;
}

public void entrar(String id){
   try {
      entrada.acquire();
      Thread.sleep(100);
      entrada.release();
   } catch (InterruptedException ex) {

      if(Character.compare(id.charAt(1),'S')==0) {
         amenaza.repeler(id);
      }else{
         amenaza.refugiar(id);
      }

   }
}

public void salir(){
   try {
      salida.acquire();
      Thread.sleep(100);
      salida.release();
   } catch (InterruptedException ex) {
      Logger.getLogger(Hormiguero.class.getName()).log(Level.SEVERE, null, ex);
   }
}

public void cogerComida(String id){
   try {
      lista.añadir(id);
      Thread.sleep(3900); //Los otros 100ms restantes son del tiempo en el tunel
      lista.quitar(id);
   } catch (InterruptedException ex) {
      Logger.getLogger(Hormiguero.class.getName()).log(Level.SEVERE, null, ex);
   }
}
```

```
    public void irZComer(String id){
        try {
            caminoZcomer.añadir(id);
            Thread.sleep(1000 + (int)(Math.random()*2000));
            caminoZcomer.quitar(id);
        } catch (InterruptedException ex) {
            Logger.getLogger(Hormiguero.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public int getObfuera() {
        return obfuera;
    }

    public int getObdentro() {
        return obdentro;
    }



}
```

Interface InterfaceColonia

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

public interface InterfaceColonia extends Remote {
    int obExterior() throws RemoteException;
    int obInterior() throws RemoteException;
    int solInstruccion() throws RemoteException;
    int solInvasion() throws RemoteException;
    int criZonaCom() throws RemoteException;
    int criRefugio() throws RemoteException;
    void causarAmenaza() throws RemoteException;
    boolean pasarAmenaza() throws RemoteException;
}
```

Clase ListaHormigas

```
import java.util.ArrayList;
import javax.swing.JTextField;

public class ListaHormigas {
    private JTextField j;
    private ArrayList<String> lista;
```

```
    public ListaHormigas(JTextField j){
        this.j = j;
        lista = new ArrayList<>();
    }


    public synchronized void añadir(String s){
        lista.add(s);
        String text = j.getText();
        j.setText(text + s + ", ");
    }

    public synchronized void quitar(String s){
        lista.remove(s);
        String text = "";
        for(int i = 0; i < lista.size(); i++){
            text+= lista.get(i);
            text+=", ";

        }
        j.setText(text);
    }

    public synchronized ArrayList<String> getLista(){
        return lista;
    }
}
```

Clase LogCreator

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;



public class LogCreator {
    private FileWriter salida;
    private Lock cerrojo = new ReentrantLock();

    public LogCreator(){
        try {
            this.salida = new FileWriter("evolucionColonia.txt");
            salida.close();
```

```
        } catch (IOException ex) {
            Logger.getLogger(LogCreator.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void añadir_aLog(String s){
        cerrojo.lock();
        try {
            Date date = new Date();
            salida = new FileWriter("evolucionColonia.txt", true);
            salida.write(date.toString()+" " + s + "\n");
            salida.close();
        } catch (IOException ex) {
            Logger.getLogger(LogCreator.class.getName()).log(Level.SEVERE, null, ex);
        }finally{cerrojo.unlock();}
    }
}
```

Clase Refugio

```
import javax.swing.JTextField;
public class Refugio {
    private ListaHormigas lista ;
    private int criaRef = 0;

    public Refugio(JTextField j) {
        lista = new ListaHormigas(j);
    }

    public synchronized void incrementarCriaR(){
        criaRef++;
    }

    public synchronized void decrementarCriaR(){
        criaRef--;
    }

    public void irRefugio(String id){
        incrementarCriaR();
        lista.añadir(id);
    }

    public void salirRefugio(String id){
        lista.quitar(id);
        decrementarCriaR();
    }

    public int getCriaRef() {
        return criaRef;
```

```
    }

}
```

Clase Servidor

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.logging.Level;
import java.util.logging.Logger;



public class Servidor extends Thread{
    private Amenaza am;
    private Hormiguero h;
    private Refugio r;
    private ZComer zc;
    private ZInstruccion zi;

    public Servidor(Amenaza am, Hormiguero h, Refugio r, ZComer zc, ZInstruccion zi){
        this.am = am;
        this.h = h;
        this.r = r;
        this.zc = zc;
        this.zi = zi;
    }

    @Override
    public void run(){

        try {
            Colonia obj = new Colonia(am,h,r,zc,zi);
            Registry registry = LocateRegistry.createRegistry(1099);
            Naming.rebind("//localhost/ObjetoColonia", obj);
        } catch (RemoteException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        } catch (MalformedURLException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Clase ZComer

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
```

```
import java.util.concurrent.locks.ReentrantLock;
import javax.swing.JTextField;


public class ZComer {
    private int comida;
    private Lock cerrojo;
    private Condition vacio;
    private ListaHormigas lista;
    private JTextField j;
    private Amenaza a;
    private int criaZcomer = 0;

    public synchronized void incrementarCriaZC(){
        criaZcomer++;
    }

    public synchronized void decrementarCriaZC(){
        criaZcomer--;
    }

    public ZComer(JTextField j1, JTextField j2, Amenaza a) {
        comida = 0;
        cerrojo = new ReentrantLock();
        vacio = cerrojo.newCondition();
        this.j = j2;
        j.setText(String.valueOf(comida));
        lista = new ListaHormigas(j1);
        this.a = a;
    }

    public void depositar(String id){
        lista.añadir(id);
        cerrojo.lock();
        try{
            Thread.sleep(1000 + (int)(Math.random()*1000));
            comida += 5;
            j.setText(String.valueOf(comida));
            vacio.signalAll();
        }catch(InterruptedException e){System.out.println("ERROR");
        }finally{cerrojo.unlock();}
        lista.quitar(id);
    }

    public void cogerComida(String id){
        lista.añadir(id);
        if(Character.compare(id.charAt(1),'C')==0) {
            incrementarCriaZC();
```

```
        }
        cerrojo.lock();
        try {
            while(comida<=0){
                vacio.await();
            }
            comida --;
            j.setText(String.valueOf(comida));
            vacio.signalAll();
        } catch (InterruptedException ex) {
            lista.quitar(id);
            if(Character.compare(id.charAt(1),'S')==0) {
                a.repeler(id);
            }else{
                decrementarCriaZC();
                a.refugiar(id);
            }
            lista.añadir(id);
        } finally {
            cerrojo.unlock();
        }

    }

    public void comer(int n, String id){
        try {
            Thread.sleep(n);
            lista.quitar(id);
            if(Character.compare(id.charAt(1),'C')==0) {
                decrementarCriaZC();
            }
        } catch (InterruptedException ex) {
            lista.quitar(id);
            if(Character.compare(id.charAt(1),'S')==0) {
                a.repeler(id);
            }else{
              decrementarCriaZC();
              a.refugiar(id);
            }
            lista.añadir(id);
        }
    }

    public int getCriaZcomer() {
        return criaZcomer;
    }
}
```

Clase ZDescanso

```
import javax.swing.JTextField;

public class ZDescanso {
    private ListaHormigas lista;
    private Amenaza a;

    public ZDescanso(JTextField j, Amenaza a) {
        this.lista = new ListaHormigas(j);
        this.a = a;
    }

    public void descansar(int n, String id){

        lista.añadir(id);
        try {
            Thread.sleep(n);
        } catch (InterruptedException ex) {
            lista.quitar(id);
            if(Character.compare(id.charAt(1),'S')==0) {
                a.repeler(id);
            }else{
                a.refugiar(id);
            }
            lista.añadir(id);
        }
        lista.quitar(id);
    }
}
```

Clase ZInstruccion

```
import javax.swing.JTextField;

public class ZInstruccion {
    private ListaHormigas lista ;
    private Amenaza a;
    private int solInst = 0;

    public synchronized void incrementarSolI(){
        solInst++;
    }

    public synchronized void decrementarSolI(){
        solInst--;
    }

    public ZInstruccion(JTextField j, Amenaza a) {
        lista = new ListaHormigas(j);
```

```
        this.a = a;
    }

    public void hacer_Instruccion(String id){
        lista.añadir(id);
        incrementarSolI();
        try {
            Thread.sleep(2000+(int)(6000*Math.random()));
            decrementarSolI();
        } catch (InterruptedException ex) {
            decrementarSolI();
            lista.quitar(id);
            a.repeler(id);
            lista.añadir(id);
        }
        lista.quitar(id);
    }

    public int getSolInst() {
        return solInst;
    }
}
```

Clase Interfaz_parte_1

```
import Logica.Almacen;
import Logica.Amenaza;
import Logica.ClaseDetener;
import Logica.CreadorHormigas;
import Logica.Hormiguero;
import Logica.LogCreator;
import Logica.Refugio;
import Logica.Servidor;
import Logica.ZComer;
import Logica.ZDescanso;
import Logica.ZInstruccion;

public class Interfaz_parte_1 extends javax.swing.JFrame {
    private ClaseDetener cd = new ClaseDetener();
    private Almacen a;
    private Hormiguero h;
    private Refugio r;
    private ZDescanso d;
    private ZInstruccion i;
    private ZComer c;
    private Amenaza am;
    private LogCreator lc;
    private Servidor s;
```

```
/**
 * Creates new form Interfaz_1
 */
public Interfaz_parte_1() {
    initComponents();
    this.lc = new LogCreator();
    this.r = new Refugio(enRefugio);
    this.h = new Hormiguero(buscando, llevando_comida, am);
    this.a = new Almacen(enAlmacen, comida_almacen);
    this.am = new Amenaza(amenaza, repeliendo,h, r, lc, txtEsp);
    this.d = new ZDescanso(descansando, am);
    this.i = new ZInstruccion(haciendo_inst, am);
    this.c = new ZComer(zona_comer, comida_zona, am);

    CreadorHormigas ch = new CreadorHormigas(a, h, r, c, d, i, cd, am, lc);
    ch.start();
    s = new Servidor(am, h, r, c, i);
    s.start();
}

/**
 * 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() {

    buscando_t = new javax.swing.JLabel();
    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();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();
    jLabel10 = new javax.swing.JLabel();
    jLabel11 = new javax.swing.JLabel();
    jLabel12 = new javax.swing.JLabel();
    buscando = new javax.swing.JTextField();
    enAlmacen = new javax.swing.JTextField();
    llevando_comida = new javax.swing.JTextField();
    haciendo_inst = new javax.swing.JTextField();
    descansando = new javax.swing.JTextField();
    comida_almacen = new javax.swing.JTextField();
```

```
        comida_zona = new javax.swing.JTextField();
        amenaza = new javax.swing.JButton();
        reanudar = new javax.swing.JButton();
        parar = new javax.swing.JButton();
        zona_comer = new javax.swing.JTextField();
        enRefugio = new javax.swing.JTextField();
        repeliendo = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        txtEsp = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        buscando_t.setText("Hormigas buscando comida");
        getContentPane().add(buscando_t, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 62, -1, -1));

        jLabel2.setText("Hormigas repeliendo un insecto invasor");
        getContentPane().add(jLabel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
111, -1, -1));

        jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jLabel3.setText("Exterior de la colonia");
        getContentPane().add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
22, -1, -1));

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jLabel4.setText("Interior de la colonia");
        getContentPane().add(jLabel4, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
148, 119, -1));

        jLabel5.setText("Hormigas en ALMACEN");
        getContentPane().add(jLabel5, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
185, -1, -1));

        jLabel6.setText("Hormigas llevando comida a la ZONA PARA COMER");
        getContentPane().add(jLabel6, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
225, -1, -1));

        jLabel7.setText("Hormigas haciendo la INSTRUCCION");
        getContentPane().add(jLabel7, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
265, -1, -1));

        jLabel8.setText("Hormigas descansando");
        getContentPane().add(jLabel8, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
305, -1, -1));

        jLabel9.setText("ZONA PARA COMER");
```

```
        getContentPane().add(jLabel9, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
367, -1, -1));

        jLabel10.setText("REFUGIO");
        getContentPane().add(jLabel10, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
407, -1, -1));

        jLabel11.setText("Comida (ALMACEN)");
        getContentPane().add(jLabel11, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
330, -1, 22));

        jLabel12.setText("Comida (ZONA PARA COMER)");
        getContentPane().add(jLabel12, new
org.netbeans.lib.awtextra.AbsoluteConstraints(530, 330, -1, -1));

        buscando.setEnabled(false);
        getContentPane().add(buscando, new
org.netbeans.lib.awtextra.AbsoluteConstraints(188, 56, 774, -1));

        enAlmacen.setEnabled(false);
        getContentPane().add(enAlmacen, new
org.netbeans.lib.awtextra.AbsoluteConstraints(164, 182, 790, -1));

        llevando_comida.setEnabled(false);
        getContentPane().add(llevando_comida, new
org.netbeans.lib.awtextra.AbsoluteConstraints(304, 222, 650, -1));

        haciendo_inst.setEnabled(false);
        getContentPane().add(haciendo_inst, new
org.netbeans.lib.awtextra.AbsoluteConstraints(233, 262, 720, -1));

        descansando.setEnabled(false);
        getContentPane().add(descansando, new
org.netbeans.lib.awtextra.AbsoluteConstraints(162, 302, 790, -1));

        comida_almacen.setEnabled(false);
        getContentPane().add(comida_almacen, new
org.netbeans.lib.awtextra.AbsoluteConstraints(145, 330, 80, -1));

        comida_zona.setEnabled(false);
        getContentPane().add(comida_zona, new
org.netbeans.lib.awtextra.AbsoluteConstraints(714, 330, 100, -1));

        amenaza.setText("Generar amenaza Insecto Invasor");
        amenaza.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                amenazaActionPerformed(evt);
            }
```

```
        });
        getContentPane().add(amenaza, new
org.netbeans.lib.awtextra.AbsoluteConstraints(304, 457, 232, -1));

        reanudar.setText("Reanudar");
        reanudar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                reanudarActionPerformed(evt);
            }
        });
        getContentPane().add(reanudar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(186, 457, -1, -1));

        parar.setText("Parar");
        parar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pararActionPerformed(evt);
            }
        });
        getContentPane().add(parar, new org.netbeans.lib.awtextra.AbsoluteConstraints(45,
457, 80, -1));

        zona_comer.setEnabled(false);
        getContentPane().add(zona_comer, new
org.netbeans.lib.awtextra.AbsoluteConstraints(147, 364, 810, -1));

        enRefugio.setEnabled(false);
        getContentPane().add(enRefugio, new
org.netbeans.lib.awtextra.AbsoluteConstraints(85, 404, 870, -1));

        repeliendo.setEnabled(false);
        getContentPane().add(repeliendo, new
org.netbeans.lib.awtextra.AbsoluteConstraints(248, 108, 710, -1));
        getContentPane().add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(520,
140, -1, -1));

        txtEsp.setText(" ");
        getContentPane().add(txtEsp, new org.netbeans.lib.awtextra.AbsoluteConstraints(530,
140, 110, 20));

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

    private void pararActionPerformed(java.awt.event.ActionEvent evt) {
        cd.detener();
        amenaza.setVisible(false);
    }
```

```
    private void reanudarActionPerformed(java.awt.event.ActionEvent evt) {
        cd.reanudar();
        amenaza.setVisible(true);
    }

    private void amenazaActionPerformed(java.awt.event.ActionEvent evt) {
        am.amenaza();
    }

    /**
     * @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(Interfaz_parte_1.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Interfaz_parte_1.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Interfaz_parte_1.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Interfaz_parte_1.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
```

```
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Interfaz_parte_1().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton amenaza;
    private javax.swing.JTextField buscando;
    private javax.swing.JLabel buscando_t;
    private javax.swing.JTextField comida_almacen;
    private javax.swing.JTextField comida_zona;
    private javax.swing.JTextField descansando;
    private javax.swing.JTextField enAlmacen;
    private javax.swing.JTextField enRefugio;
    private javax.swing.JTextField haciendo_inst;
    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.JTextField llevando_comida;
    private javax.swing.JButton parar;
    private javax.swing.JButton reanudar;
    private javax.swing.JTextField repeliendo;
    private javax.swing.JLabel txtEsp;
    private javax.swing.JTextField zona_comer;
    // End of variables declaration
}
```

Clase Interfaz_parte_2

```
import Logica.Cliente;

public class Interfaz_parte_2 extends javax.swing.JFrame {
    private Cliente c;
    /**
     * Creates new form Interfaz_2
     */
```

```
public Interfaz_parte_2() {
    initComponents();
    c = new Cliente(extCol, intCol, hacInst, repeliendo, criaCom, criaRef, botAmenaza);
    c.start();
}

/**
 * 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() {

    jTextField2 = new javax.swing.JTextField();
    jLabel1 = new javax.swing.JLabel();
    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();
    botAmenaza = new javax.swing.JButton();
    extCol = new javax.swing.JTextField();
    hacInst = new javax.swing.JTextField();
    repeliendo = new javax.swing.JTextField();
    criaCom = new javax.swing.JTextField();
    intCol = new javax.swing.JTextField();
    criaRef = new javax.swing.JTextField();

    jTextField2.setEditable(false);
    jTextField2.setEnabled(false);

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jLabel1.setText("Numeros de hormigas obreras en el exterior de la colonia");

    jLabel2.setText("Numero de hormigas obreras en el interior de la colonia");

    jLabel3.setText("Numero de hormigas sodado haciendo la instruccion");

    jLabel4.setText("Numero de hormigas soldado repeliendo el invasor");

    jLabel5.setText("Numero de hormigas cria en la ZONA PARA COMER");

    jLabel6.setText("Numero de hormigas cria en el REFUGIO");

    botAmenaza.setText("Generar Amenaza Insecto Invasor");
```

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

        extCol.setEditable(false);
        extCol.setEnabled(false);

        hacInst.setEditable(false);
        hacInst.setEnabled(false);

        repeliendo.setEditable(false);
        repeliendo.setEnabled(false);

        criaCom.setEditable(false);
        criaCom.setEnabled(false);

        intCol.setEditable(false);
        intCol.setEnabled(false);

        criaRef.setEditable(false);
        criaRef.setEnabled(false);

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

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(175, 175, 175)
                        .addComponent(botAmenaza))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(30, 30, 30)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel1)
                            .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE,
306, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE,
306, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE,
276, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel5)
                            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE,
276, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                    .addGap(81, 81, 81)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(criaRef, javax.swing.GroupLayout.PREFERRED_SIZE,
71, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(criaCom, javax.swing.GroupLayout.PREFERRED_SIZE,
71, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(repeliendo,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(hacInst, javax.swing.GroupLayout.PREFERRED_SIZE,
71, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(extCol, javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(intCol, javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addContainerGap(88, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(95, 95, 95)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(extCol, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(intCol, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(hacInst, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel4)
                    .addComponent(repeliendo, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

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

```
                .addComponent(jLabel5)
                .addComponent(criaCom, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel6)
                .addComponent(criaRef, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 102,
Short.MAX_VALUE)
            .addComponent(botAmenaza)
            .addGap(65, 65, 65))
        );

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

    private void botAmenazaActionPerformed(java.awt.event.ActionEvent evt) {
        c.setAmenaza(true);
    }

    /**
     * @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(Interfaz_parte_2.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);
        } catch (InstantiationException ex) {
```

```
java.util.logging.Logger.getLogger(Interfaz_parte_2.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Interfaz_parte_2.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Interfaz_parte_2.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Interfaz_parte_2().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton botAmenaza;
    private javax.swing.JTextField criaCom;
    private javax.swing.JTextField criaRef;
    private javax.swing.JTextField extCol;
    private javax.swing.JTextField hacInst;
    private javax.swing.JTextField intCol;
    private javax.swing.JLabel jLabel1;
    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.JTextField jTextField2;
    private javax.swing.JTextField repeliendo;
    // End of variables declaration
}
```