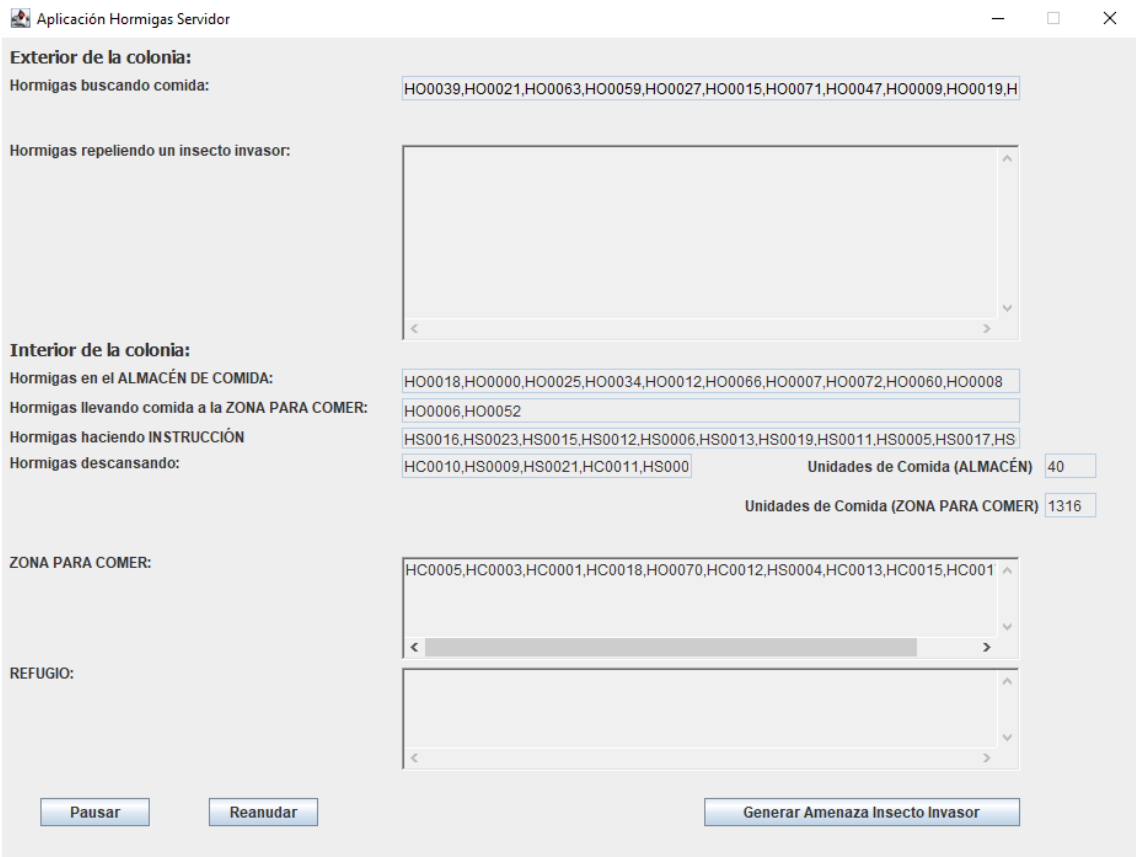
PECL - Programación Avanzada

Simulación del funcionamiento de una colonia de hormigas



Convocatoria Ordinaria Curso 2022-2023 Grado de Ingeniería de Computadores

Creado por:

- Emilio Macías Do Santos

Contenido

Análisis de alto nivel (descripción general del problema e identificación de sus principales actores).

Para la realización de la práctica se han seguido los conceptos del paradigma de programación orientado a objetos, permitiendo una estructuración del código de forma efectiva gracias a la abstracción de los elementos involucrados, su independencia con respecto a otras partes del programa gracias al encapsulamiento; y usando los principios de herencia y polimorfismo se ha conseguido seguir las exigencias del proyecto.

Para ello, se ha optado por una lógica Modelo-Vista-Controlador (MVC), debido a su versatilidad y a la independencia de sus componentes. Permitiendo reutilizar clases sin duplicarlas gracias al uso de clases heredadas e interfaces.

Este concepto es de suma importancia en el proyecto, ya que gracias a esta lógica se ha podido realizar la segunda parte de la práctica fácilmente.

- En la primera parte de la práctica, se ha programado usando MVC para poder realizar la programación concurrente separando la parte del cómputo o comportamiento de las hormigas del resto del programa; como su visualización, siendo así independiente. Correspondiendo esta lógica con los siguientes componentes implicados:
    - Se puede considerar el Controlador como el encargado de definir el comportamiento general de cada una de las hormigas, de cómo interactúan unas con otras, y de proporcionar los recursos compartidos que usarán durante todo su ciclo de vida.
    - El Modelo es la representación en programación orientada a objetos de las entidades presentes en el programa, en este caso los diferentes tipos de hormigas y su ciclo de vida a lo largo del programa. Para realizar dicha representación de forma efectiva se ha optado por el uso de una interfaz Hormiga que defina un comportamiento genérico y común a todas.
    - La vista es la encargada de mostrarnos la información y de interactuar con el usuario. Permitiendo parar la ejecución del programa, retomar dicha ejecución y generar un insecto invasor que modifique el comportamiento temporalmente de algunas hormigas.
- En la segunda parte de la práctica, se ha programado usando RMI debido a su facilidad de uso al haber implementado la primera parte usando la lógica MVC. Con ello, se ha conseguido suplir el requisito de la programación distribuida permitiendo realizar peticiones de datos sobre el comportamiento de las hormigas, y su correspondiente respuesta en tiempo real separando las responsabilidades de la aplicación cliente

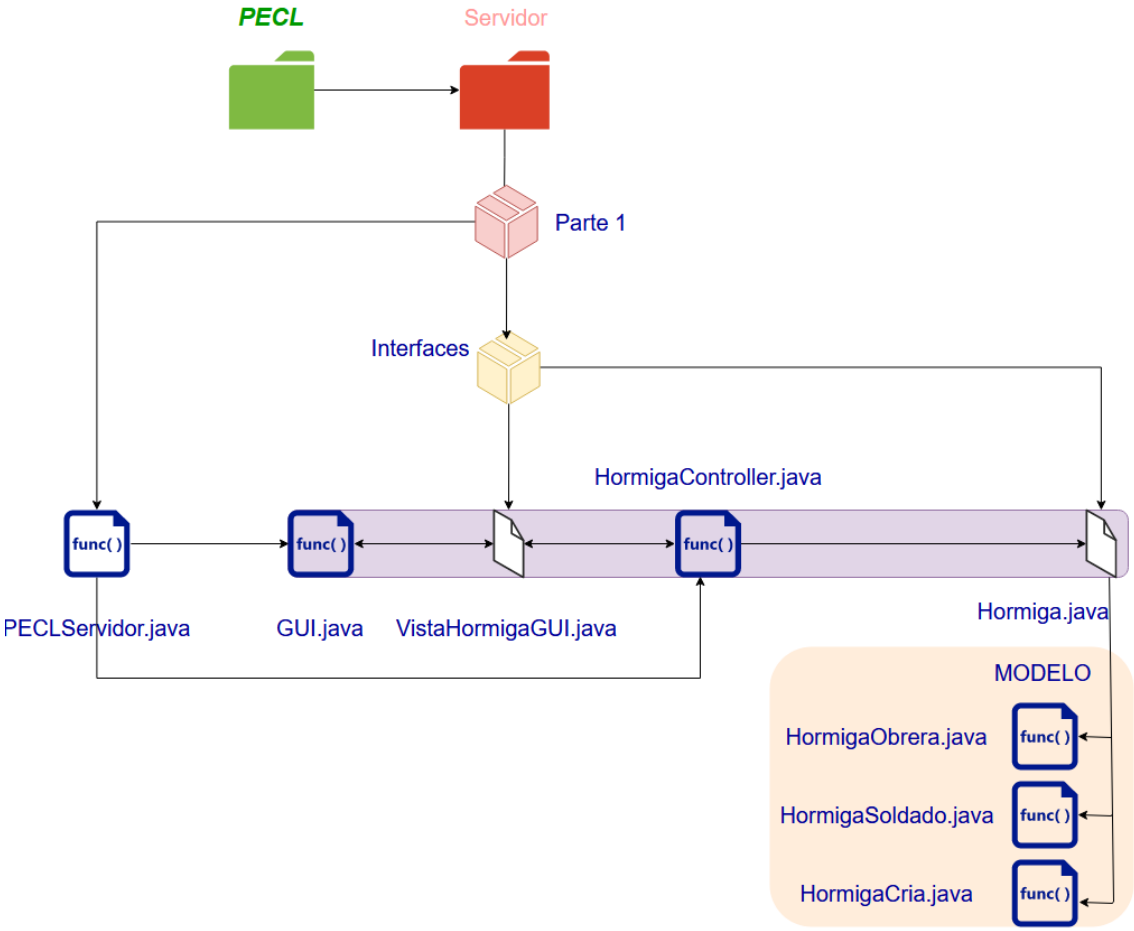
(encargada de obtener los datos de las hormigas y generar un insecto invasor), y la del servidor (procesar el comportamiento de todas las hormigas y gestionar la información de la colonia). También permite que múltiples aplicaciones cliente se conecten para obtener dicha información, y el servidor de forma efectiva procesa las peticiones.

Para resumir, se pueden considerar los objetos de los modelos implicados como las hormigas presentes en la colonia, las clases implicadas como la representación de su tipo, y el controlador como el hormiguero o colonia que gestiona cada una de las entidades presentes; permitiendo una simulación efectiva.

Diseño general del sistema y discusión de las herramientas de sincronización utilizadas.

Diseño primera parte

Para la primera parte de la práctica, se ha optado por el siguiente diseño Modelo-Vista-Controlador (MVC):



Como se puede observar se tienen los siguientes elementos implicados:

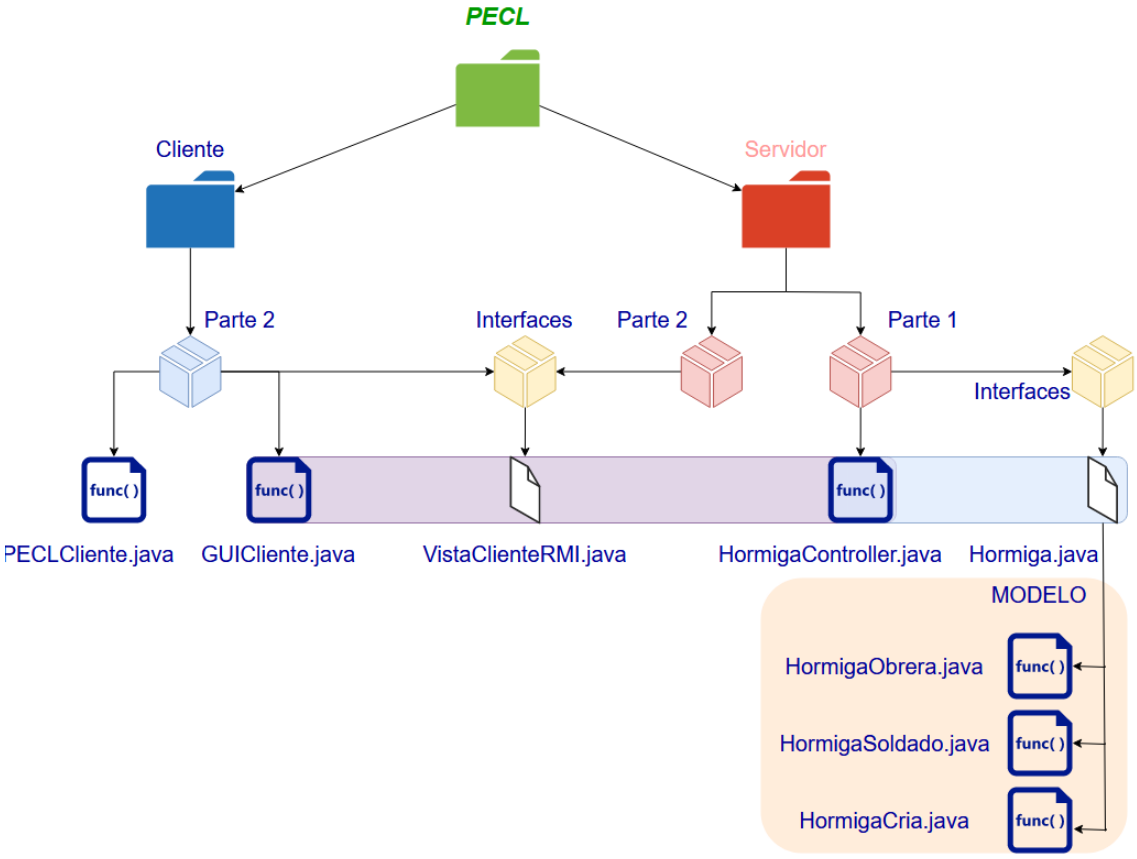
- “PECLServidor.java”:
    - Dentro del paquete “parte1”
    - Se encarga de inicializar el controlador y la interfaz gráfica.
- “GUI.java”
    - Vista del MVC.
    - Dentro del paquete “parte1”
    - Utiliza la interfaz “VistaHormigaGUI.java” para comunicarse con el controlador.
    - Muestra todos los elementos gráficos de la aplicación servidor, obteniendo los datos en tiempo real del comportamiento del hormiguero (gracias al controlador), y permite enviar información a éste para modificar el comportamiento de algunas hormigas.

- “VistaHormigaGUI.java”
    - Interfaz que expone los métodos que utilizará la vista o vistas (el MVC debido a su abstracción permite meter más de una vista fácilmente) para comunicarse con el controlador.
    - Presente en el paquete “parte1.interfaces”
    - Se puede implementar para crear varias vistas comunicándose simultáneamente con el controlador.
- “HormigaController.java”
    - Es la clase más importante del proyecto.
    - Presente en el paquete “parte1”
    - Permite realizar el cómputo del comportamiento de las hormigas, cómo se relacionan entre ellas y gestionar los recursos compartidos de forma efectiva.
    - Utiliza la interfaz “Hormiga.java”
    - Utiliza los modelos para proporcionar una lógica de entidades interrelacionadas con un ciclo de vida y un comportamiento común según el tipo de hormiga.
    - Sin esta clase, el programa no podría funcionar, a diferencia de las vistas que son opcionales.
- “Hormiga.java”
    - Es una interfaz que define un comportamiento común entre modelos de Hormiga para permitir la definición de acciones genéricas asociadas a una entidad abstracta Hormiga.
    - Presente en el paquete “parte1.interfaces”
    - Permite definir el resto de modelos de tipo Hormiga.
- Modelos de tipo Hormiga:
    - Permiten que el controlador pueda desempeñar una lógica basada en estas entidades.

- Los modelos definen el ciclo de vida que seguirá una hormiga. No su comportamiento por tipo. Esto tendrá relevancia y se explicará más adelante.
- “HormigaObrera.java”
  - Representa el comportamiento que tendrán todas las hormigas obreras.
- “HormigaSoldado.java”
  - Representa el comportamiento que tendrán todas las hormigas soldado.
- “HormigaCria.java”
  - Representa el comportamiento que tendrán todas las hormigas cría.

Diseño segunda parte

Para la segunda parte de la práctica, necesitamos realizar un sistema con programación distribuida. Gracias al MVC seguido al realizar el trabajo, se ha podido realizar el siguiente esquema del proyecto:



Como se puede observar, existen dos aplicaciones independientes:

- Aplicación Cliente:
  - Interfaz "VistaClienteRMI.java"
    - Parte del paquete "parte2.interfaces".
    - Usado por ambas aplicaciones (cliente y servidor).
    - Se ha conectado la clase GUICliente.java con el controlador usando esta interfaz.
    - Permite usar RMI para las comunicaciones.
  - GUICliente.java
    - Vista de la aplicación cliente.
    - Sirve para mostrar la información.
    - Hace uso de la anterior interfaz para comunicarse con el controlador mediante RMI.
  - PECLCliente.java
    - Clase destinada a ejecutar la aplicación.

- Aplicación Servidor:
    - Interfaz “VistaClienteRMI.java”
        - Parte del paquete “parte2.interfaces”.
        - Usado por ambas aplicaciones (cliente y servidor).
        - Se ha conectado la clase GUICliente.java con el controlador usando esta interfaz.
        - Permite usar RMI para las comunicaciones.
    - HormigaController.java
        - Parte del paquete “parte1”.
        - La clase más importante del proyecto.
        - Mediante el uso de las interfaces del paquete “parte1.interfaces”, interconecta la lógica MVC.
        - Permite abstraer las comunicaciones recibidas por RMI de la vista de la aplicación cliente fácilmente.
        - Más adelante se explicará con detalle esta clase.
    - Interfaz “Hormiga.java”
        - Parte del paquete “parte1.interfaces”
        - Mediante el uso de esta interfaz, permite definir en el controlador un comportamiento común a todas las hormigas.
        - Usado por las clases del modelo para implementar una lógica MVC efectiva.
    - Clases Modelo
        - Al implementar la interfaz “Hormiga.java” permite especificar comportamientos específicos según el tipo de hormiga.
        - Tipos:
            - HormigaObrera.java
            - HormigaSoldado.java
            - HormigaCria.java
        - Al programarlo siguiendo el paradigma de programación de objetos, se puede obtener una abstracción efectiva de las entidades modelo involucradas en la ejecución del programa.

Habiendo descrito superficialmente los elementos implicados en la lógica del programa y su estructura, a continuación, se procederá a explicar los mecanismos utilizados para desempeñar correctamente la función de cada uno de los elementos implicados en el programa.

Discusión de las herramientas de sincronización utilizadas

Semáforos

Se han utilizado numerosos semáforos para permitir gestionar los recursos compartidos de forma eficiente sin realizar espera activa.

Tipos de semáforos utilizados:

- Semáforos de sincronización:
  - Usados para gestionar un recuso compartido, por ejemplo notificar o desbloquear a otro hilo cuando haya un recurso disponible.

```
private Semaphore semUnidadesAlmacenVacio;
private Semaphore semUnidadesAlmacenExclusion;
private Semaphore semUnidadesAlmacenLleno;
```

```
this.semUnidadesAlmacenVacio = new Semaphore(0,true);
this.semUnidadesAlmacenExclusion = new Semaphore(1,true);
this.semUnidadesAlmacenLleno = new Semaphore(this.semAlmacenComida.availablePermits(),true);
```

```
public void entrarAlmacenComidaDepositar(HormigaObrera hormiga) {
    try {
        this.semUnidadesAlmacenLleno.acquire();
        this.semAlmacenComida.acquire(); // Entra en el almacen de comida
        // Está la hormiga en el tunel
        this.addLog(hormiga, ", ha entrado al almacén de comida a depositar.");
        this.hAlmacenComida.add(hormiga.getName()); // Queda reflejado que está dentro del almacen
        this.cambioRelease();
        Thread.sleep((long) (((Math.random()*2)+2)*1000)); // Tiempo que tarda en depositarlo
        this.semUnidadesAlmacenExclusion.acquire(); // Exclusion
        this.unidadesComidaAlmacen+=5; // Deposita la comida en el almacén
        this.semUnidadesAlmacenExclusion.release(); // Exclusion
        this.cambioRelease();
        this.addLog(hormiga, ", ha depositado la comida y ha salido del almacén.");
    } catch(InterruptedException ie) {}
    finally {
        this.semAlmacenComida.release();
        this.semUnidadesAlmacenVacio.release(); // Sincronizacion
        this.hAlmacenComida.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

  - En este ejemplo se usan los semáforos para la sincronización (notificación) de cuando se inserta un elemento en el almacén, para que las hormigas encargadas lo saquen si se encuentran a la espera.

- Semáforos de exclusión mutua:
    - Usados para que solamente una hormiga acceda a un recurso compartido.

```
// Semáforos para los tuneles
private Semaphore semTunelEntrada;
private Semaphore semTunelSalida;
```

```
// Semáforos para los túneles
this.semTunelEntrada = new Semaphore(1,true);
this.semTunelSalida = new Semaphore(2,true);
```

```
//  Métodos para los túneles
public void pasarTunelEntrada(Hormiga hormiga) {
    try {
        this.semTunelEntrada.acquire();
        this.hFueraColonia.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
        // Está la hormiga en el tunel
        this.addLog(hormiga, ", ha pasado por un tunel de entrada.");
        Thread.sleep(100);
        this.addLog(hormiga, ", ha salido por un tunel de entrada.");
    } catch(InterruptedException ie) {}
    finally {
        this.semTunelEntrada.release();
        this.hDentroColonia.add(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

- En este ejemplo se realiza una exclusión mutua para las hormigas que intentan acceder a un túnel de entrada, ya que solamente permite transitar una hormiga por túnel.

CopyOnWriteArraySet

Usado para guardar las hormigas que se encuentran accediendo a un recurso determinado como por ejemplo:

```
// Recursos compartidos para saber donde estás las hormigas en cada zona
private Set<String> hDentroColonia;
private Set<String> hFueraColonia;
```

```
// Listas de Zonas
this.hDentroColonia = new CopyOnWriteArraySet<String>();
this.hFueraColonia = new CopyOnWriteArraySet<String>();
```

```
public void pasarTunelSalida(Hormiga hormiga) {
    try {
        this.semTunelSalida.acquire();
        this.hDentroColonia.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
        // Está la hormiga en el tunel
        this.addLog(hormiga, ", ha pasado por un tunel de salida.");
        Thread.sleep(100);
        this.addLog(hormiga, ", ha salido por un tunel de salida.");
    } catch(InterruptedException ie) {}
    finally {
        this.semTunelSalida.release();
        this.hFueraColonia.add(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

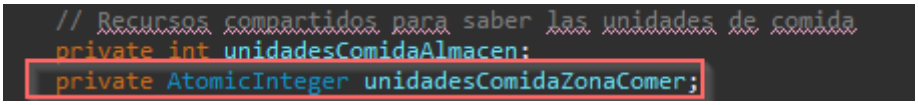
De esta forma se consigue gestionar qué están haciendo las hormigas en cada instante de tiempo.

Se ha optado por un CopyOnWriteArraySet debido a:

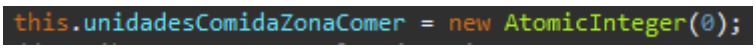
- No pueden haber dos hormigas en dos zonas a la vez, por lo que no hay duplicidad (Set).
- Permite almacenar las hormigas en una estructura de fácil acceso como una lista enlazada.
- Permite realizar modificaciones solamente por una hormiga a la vez.
- Debido a que se va ha utilizar una estructura de MVC con posibilidad de múltiples vistas que consulten información simultánea (usando programación distribuida con RMI) es posible que se hagan muchas lecturas en un mismo instante de tiempo.

Clases Atomic:

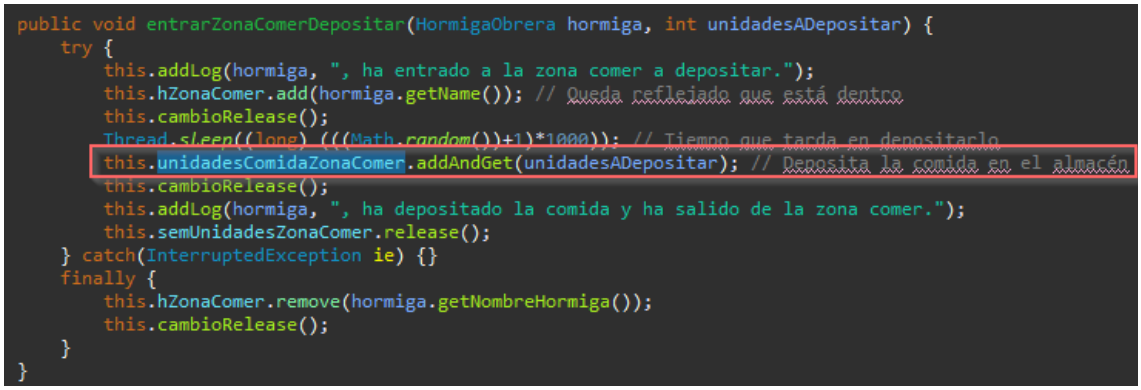
- AtomicInteger:
  - Se ha utilizado para que se gestione de forma efectiva las unidades de alimento presentes en Zona Comer.
  - Permiten que solamente un hilo modifique a la vez el objeto, y su lectura sin condiciones de carrera.



```
// Recursos compartidos para saber las unidades de comida
private int unidadesComidaAlmacen;
private AtomicInteger unidadesComidaZonaComer;
```

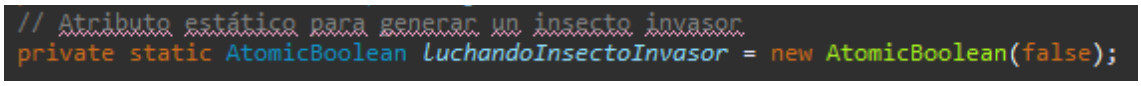


```
this.unidadesComidaZonaComer = new AtomicInteger(0);
```



```
public void entrarZonaComerDepositar(HormigaObrera hormiga, int unidadesADepositar) {
    try {
        this.addLog(hormiga, ", ha entrado a la zona comer a depositar.");
        this.hZonaComer.add(hormiga.getName()); // Queda reflejado que está dentro
        this.cambioRelease();
        Thread.sleep((long) (((Math.random())+1)*1000)); // Tiempo que tarda en depositarlo
        this.unidadesComidaZonaComer.addAndGet(unidadesADepositar); // Deposita la comida en el almacén
        this.cambioRelease();
        this.addLog(hormiga, ", ha depositado la comida y ha salido de la zona comer.");
        this.semUnidadesZonaComer.release();
    } catch(InterruptedException ie) {}
    finally {
        this.hZonaComer.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

- AtomicBoolean:
  - Se ha utilizado para saber si se ha pausado la ejecución del programa, o si las hormigas soldado están luchando contra el insecto invasor.



```
// Atributo estático para generar un insecto invasor
private static AtomicBoolean luchandoInsectoInvasor = new AtomicBoolean(false);
```

```java
@Override
public void generarInsectoInvasor() {
    if(luchandoInsectoInvasor.getAndSet(true) == false) {
        if(this.pararEjecucion.get()) {
            luchandoInsectoInvasor.set(false);
            return;
        }
        this.semLuchandoInsecto.drainPermits();
        hormigasSoldado.interrupt();
        hormigasCria.interrupt();
        new Thread(Thread.currentThread().getThreadGroup(),
            new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(20*1000);
                    } catch (InterruptedException e) {}
                    finally {
                        luchandoInsectoInvasor.set(false);
                        semLuchandoInsecto.release(10000);
                    }
                }
        }, "HiloControlInsectoInvasor").start();;
    }
}
```

BlockingQueue

Permite que la clase Logging.java pueda obtener las peticiones de log recibidas por el controlador para guardar en disco el comportamiento de las hormigas.

- Los hilos hormiga que quieran escribir su comportamiento, generarán un objeto de tipo String indicando la acción que están desempeñando actualmente.
- El hilo del Logging se encargará de ir escribiendo estas acciones en el disco para permitir un sistema de logging efectivo sin esperas activas.

```java
public HormigaController() throws RemoteException{
    // Me creo la cola del Logging
    this.log = new Logging(new LinkedBlockingQueue<String>());
```

```java
// Métodos
public void addLog(Hormiga hormiga, String mensaje) throws InterruptedException {
    log.getListaLog().put(Logging.getMensaje("La hormiga "+ hormiga.getNombreHormiga() + mensaje));
}
```

```java
public void transicionZonaComer(HormigaObrera hormiga) {
    try {
        this.addLog(hormiga, ", está haciendo el camino para la zona para comer.");
        this.hLlevandoZonaComer.add(hormiga.getName()); // Queda reflejado que esta dentro del almacen
        this.cambioRelease();
        Thread.sleep((long) (((Math.random()*2)+1)*1000)); // Tiempo que tarda en depositarlo
        this.addLog(hormiga, ", ha llegado a la zona comer.");
    } catch(InterruptedException ie) {}
    finally {
        this.hLlevandoZonaComer.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

```java
@Override
public void run() {
    try (FileWriter out = new FileWriter(nombreFichero)) {
        while(true) {
            out.write(listaLog.take()+"\n");
            out.flush();
        }
    } catch (InterruptedException | IOException e) {e.printStackTrace();}
}
```

- Las hormigas accederán a los recursos compartidos e irán notificando, utilizando el controlador de sus acciones para su depuración y control.

Descripción de las clases principales (atributos y métodos).

Aplicación Servidor

Logging

Permite crear un hilo independiente que se encargue de procesar las solicitudes de escritura de los eventos que suceden durante el transcurso del programa.

Se ha configurado una excepción personalizada denominada LoggingException para informar al programador cuando se produzca algún error relacionado con el logging del programa.

```
public class Logging implements Runnable{

    public static class LoggingException extends Exception{
        private static final long serialVersionUID = -3604260959065292002L;

        LoggingException(){
            super("No se ha configurado la cola del logging y es obligatoria para la ejecución de los hilos.");
        }

        LoggingException(String mensaje){
            super(mensaje);
        }
    }
```

(El resto del comportamiento de la clase se ha explicado anteriormente y se omite)

PECLServidor

Configura el controlador, permite exponerlo públicamente usando RMI y genera la interfaz gráfica. Es la clase Main que inicia el servidor.

```
// Inicializo el controlador
HormigaController controlador = new HormigaController();
new Thread(new ThreadGroup("HilosControladores"), controlador).start();
//Arranca el espacio de nombres RMIRegistry local en el puerto 1099. No es obligatorio
Registry registry = LocateRegistry.createRegistry(1099);
//Hace visible el objeto para clientes
Naming.rebind("//127.0.0.1/ControladorHormiga", controlador);
```

(Se omite la creación de la interfaz gráfica por ser poco relevante)

GUI

Carga los componentes gráficos y utilizando la interfaz VistaHormigaGUI permite obtener información del controlador y llamar a sus métodos.

```
public class GUI {

    private VistaHormigaGUI controlador;    // Usado para la logica MVC

    private JFrame frmAplicacinHormigas;
    private JTextField txtfHormigasBuscandoComida;
    private JTextField txtfHormigasAlmacenComida;
    private JTextField txtfHomigasLlevandoZonaComer;
```

Se crea un hilo nuevo para recargar en tiempo real los elementos de la interfaz gráfica. Se utiliza un semáforo para que solamente se recargue cuando se actualicen los recursos y no se produzca espera activa.

```
protected void crearHiloCambios(GUI gui) {
    new Thread(Thread.currentThread().getThreadGroup(),
            new Runnable() {
                public void run() {
                    try {
                        while(true) {
                            try {
                                gui.controlador.cambioAdquire();
                            } catch (InterruptedException ie) {}
                            gui.getTxtfUnidadesAlmacen().setText(Integer.toString(gui.controlador.getUnidadesComidaAlmacen()));
                            gui.getTxtfUnidadesZonaComer().setText(gui.controlador.getUnidadesComidaZonaComer().toString());
                            gui.getTxtfHormigasBuscandoComida().setText(String.join(",", gui.controlador.gethBuscandoComida()));
                            gui.getTxtaHormigasRepeliendoInsecto().setText(String.join(",", gui.controlador.gethRepeliendoInsecto()));
                            gui.getTxtfHormigasAlmacenComida().setText(String.join(",", gui.controlador.gethAlmacenComida()));
                            gui.getTxtfHomigasLlevandoZonaComer().setText(String.join(",", gui.controlador.gethLlevandoZonaComer()));
                            gui.getTxtfHomigasInstruccion().setText(String.join(",", gui.controlador.gethZonaInstruccion()));
                            gui.getTxtfHomigasDescansando().setText(String.join(",", gui.controlador.gethZonaDescanso()));
                            gui.getTxtaZonaParaComer().setText(String.join(",", gui.controlador.gethZonaComer()));
                            gui.getTxtaRefugio().setText(String.join(",", gui.controlador.gethRefugio()));
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            },
            "HiloEventoControlador"
        ).start();
}
```

Se han creado ActionListener en los botones para pausar, retomar o generar un insecto invasor.

```
btnPausar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        controlador.pararEjecucion();
        btnGenerarAmenazaInsecto.setEnabled(false);
    }
});
```

```
btnReanudar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        controlador.continuarEjecucion();
        btnGenerarAmenazaInsecto.setEnabled(true);
    }
});
```

```
btnGenerarAmenazaInsecto.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        controlador.generarInsectoInvasor();
    }
});
```

VistaHormigaGUI

Gestiona los datos que accederá la vista del servidor para representar visualmente el comportamiento de las hormigas. También permite usar los métodos explicados anteriormente.

```
// Métodos que utilizan un semáforo para controlar los eventos.
public void cambioAdquire() throws InterruptedException;

public void cambioRelease();
```

- Usando este método el hilo encargado de actualizar los componentes sabrá si se ha modificado algún recurso compartido.

Interfaz Hormiga

Permite definir el comportamiento común entre todas las hormigas, como por ejemplo para acceder a un recurso compartido. Es parte de la definición del modelo, lo implementarán el resto de las clases modelo.

```
public interface Hormiga {

    // Atributos comunes a todas las hormigas.
    public String getIdType();
    public int getIdHormiga();
```

Hormiga Obrera

Permite definir el ciclo de vida de las hormigas obreras.

```
public class HormigaObrera extends Thread implements Hormiga{

    // Define sus propios valores de clase
    protected static String idType = "HO";
    protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHO");
    protected static AtomicInteger contador = new AtomicInteger(0);

    static {
        HormigaController.setHormigasObreras(grupoHilos);
    }

    // Atributos especificos de cada hormiga obrera
    private int idHormiga;
    private HormigaController controlador;
```

Ejemplo de comportamiento de una hormiga obrera par:

```
if(this.getIdHormiga() % 2 == 0) { // Hormiga obrera par
    this.controlador.entrarAlmacenComidaConsumir(this);
    this.controlador.transicionZonaComer(this);
    this.controlador.entrarZonaComerDepositar(this, 5);
}
```

Como se puede observar todo su comportamiento se encuentra definido en el controlador, este modelo solamente define su ciclo de ejecución.

Si se ha seleccionado el botón de parar en la GUI, se suspenderá la ejecución de la hormiga:

```
if(this.controlador.getPararEjecucion().get()) { // Si está true, se suspende la hormiga.
    this.suspend();
}
```

(Este comportamiento se utiliza en todas las hormigas, se omitirá en las demás)

Hormiga Cría

Se realizan llamas al controlador para definir su comportamiento.

```
public class HormigaCria extends Thread implements Hormiga{

    // Define sus propios valores de clase
    protected static String idType = "HC";
    protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHC");
    protected static AtomicInteger contador = new AtomicInteger(0);

    static {
        HormigaController.setHormigasCria(grupoHilos);
    }

    // Atributos especificos de cada hormiga obrera
    private int idHormiga;
    private HormigaController controlador;
```

Si hay un insecto invasor se irá al refugio:

```
if(HormigaController.getLuchandoInsectoInvasor().get()) {
    this.controlador.entrarZonaRefugio(this);
}
```

```
@Override
public String getNombreHormiga() {
    return idType.concat(String.format("%04d", this.idHormiga));
}
```

(Método común a todas las hormigas, se omitirá en las demás)

Hormiga Soldado

```
public class HormigaSoldado extends Thread implements Hormiga{

    // Define sus propios valores de clase
    protected static String idType = "HS";
    protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHS");
    protected static AtomicInteger contador = new AtomicInteger(0);

    static {
        HormigaController.setHormigasSoldado(grupoHilos);
    }

    // Atributos especificos de cada hormiga obrera
    private int idHormiga;
    private HormigaController controlador;
```

Ejemplo de ejecución sin ataque:

```
this.controlador.entrarZonaInstruccion(this);
this.controlador.entrarZonaDescanso(this, 2);
```

Al recibir un ataque la colonia se producirá una interrupción al grupo de hilos “HilosHS” y se prepararán para la guerra. (Definido en el controlador)

Hormiga Controller

Preparada para escuchar peticiones de red provenientes de aplicaciones cliente.

```
public class HormigaController extends UnicastRemoteObject implements Runnable, VistaHormigaGUI, VistaClienteRMI {
    /**
     *
     */
    private static final long serialVersionUID = 7811150400089455866L;
    // Semáforos para los tuneles
    private Semaphore semTunelEntrada;
    private Semaphore semTunelSalida;
    // Semáforos para las zonas compartidas:
    private Semaphore semAlmacenComida;
    private Semaphore semUnidadesAlmacenVacio;
    private Semaphore semUnidadesAlmacenExclusion;
    private Semaphore semUnidadesAlmacenLleno;
    private Semaphore semUnidadesZonaComer;
    // Semáforo para notificar a las hormigas cria
    private Semaphore semLuchandoInsecto;
    // Atributos
    private Logging log;
```

Tiene numerosos semáforos para gestionar los recursos de forma efectiva.

Recursos compartidos:

```
// Recursos compartidos para saber donde estás las hormigas en cada zona
private Set<String> hDentroColonia;
private Set<String> hFueraColonia;
private Set<String> hBuscandoComida;
private Set<String> hRepeliendoInsecto;
private Set<String> hAlmacenComida;
private Set<String> hLlevandoZonaComer;
private Set<String> hZonaInstruccion;
private Set<String> hZonaDescanso;
private Set<String> hZonaComer;
private Set<String> hRefugio;
```

Grupos de hilos de hormiga (para poder interrumpirlas o despertarlas):

```
// Atributos estáticos para obtener los grupos de hilos de las hormigas
private static ThreadGroup hormigasObreras;
private static ThreadGroup hormigasSoldado;
private static ThreadGroup hormigasCria;
```

Método específico de una hormiga obrera:

```
public void entrarAlmacenComidaDepositar(HormigaObrera hormiga) {
    try {
        this.semUnidadesAlmacenLleno.acquire();
        this.semAlmacenComida.acquire(); // Entra en el almacen de comida
        // Está la hormiga en el tunel
        this.addLog(hormiga, ", ha entrado al almacén de comida a depositar.");
        this.hAlmacenComida.add(hormiga.getName()); // Queda reflejado que está dentro del almacen
        this.cambioRelease();
        Thread.sleep((long) (((Math.random()*2)+2)*1000)); // Tiempo que tarda en depositarlo
        this.semUnidadesAlmacenExclusion.acquire(); // Exclusion
        this.unidadesComidaAlmacen+=5; // Deposita la comida en el almacén
        this.semUnidadesAlmacenExclusion.release(); // Exclusion
        this.cambioRelease();
        this.addLog(hormiga, ", ha depositado la comida y ha salido del almacén.");
    } catch(InterruptedException ie) {}
    finally {
        this.semAlmacenComida.release();
        this.semUnidadesAlmacenVacio.release(); // Sincronizacion
        this.hAlmacenComida.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

Método específico de una hormiga soldado:

```
// Métodos hormigas soldado
public void entrarZonaInstruccion(HormigaSoldado hormiga) {
    try {
        this.addLog(hormiga, ", ha entrado a la zona instruccion a entrenar.");
        this.hZonaInstruccion.add(hormiga.getName()); // Queda reflejado que está dentro
        this.cambioRelease();
        Thread.sleep((long) (((Math.random()*6)+2)*1000)); // Tiempo que tarda en entrenar
        this.addLog(hormiga, ", ha terminado de entrenar y ha salido de la zona de instrucción.");
        this.hZonaInstruccion.remove(hormiga.getNombreHormiga());
    } catch(InterruptedException ie) {
        this.hZonaInstruccion.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
        this.lucharInsectoInvasor(hormiga);
    }
    finally {
        this.cambioRelease();
    }
}
```

Método específico de una hormiga cría:

```
// Métodos hormigas cria
public void entrarZonaRefugio(HormigaCria hormiga) {
    this.hRefugio.add(hormiga.getNombreHormiga());
    this.cambioRelease();
    try {
        this.addLog(hormiga, ", ha entrado al refugio (sólo hormigas cria)."); // Entra en el refugio
        this.semLuchandoInsecto.acquire();
        this.addLog(hormiga, ", ha salido del refugio (sólo hormigas cria).");
    } catch (InterruptedException e) {}
    finally {
        this.hRefugio.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

Método para todas las hormigas:

```
public void entrarZonaDescanso(Hormiga hormiga, int tiempoDescansando) {
    try {
        this.addLog(hormiga, ", ha entrado a la zona de descanso a mimir.");
        this.hZonaDescanso.add(hormiga.getNombreHormiga()); // Queda reflejado que está dentro
        this.cambioRelease();
        Thread.sleep(tiempoDescansando * 1000); // Tiempo que tarda en comer
        this.addLog(hormiga, ", ha descansado y ha salido de la zona de descanso.");
    } catch(InterruptedException ie) {
        this.hZonaDescanso.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
        if(hormiga.getIdType() == "HS") { // Es una hormiga soldado a luchar
            this.lucharInsectoInvasor((HormigaSoldado) hormiga);
        }
        if(hormiga.getIdType() == "HC") { // Es una hormiga cria al refugio
            this.entrarZonaRefugio((HormigaCria) hormiga);
        }
    }
    finally {
        this.hZonaDescanso.remove(hormiga.getNombreHormiga());
        this.cambioRelease();
    }
}
```

VistaClienteRMI

Definirá los métodos que se podrán acceder desde las aplicaciones cliente como por ejemplo:

```
public interface VistaClienteRMI extends Remote{
    // Getter y Setters públicos
    public Set<String> gethDentroColoniaRMI() throws RemoteException;

    public Set<String> gethFueraColoniaRMI() throws RemoteException;
```

(Son wrappers de los métodos ya definidos en la interfaz VistaHormigaGUI pero adaptadas a la red)

Aplicación Cliente

PECLCliente

Clase main que inicializa el controlador mediante peticiones RMI usando la interfaz del siguiente apartado:

```
VistaClienteRMI controlador = (VistaClienteRMI) Naming.lookup("//127.0.0.1/ControladorHormiga");
```

Posteriormente inicializa la interfaz gráfica GUI Cliente.

Interfaz VistaClienteRMI

Explicada anteriormente.

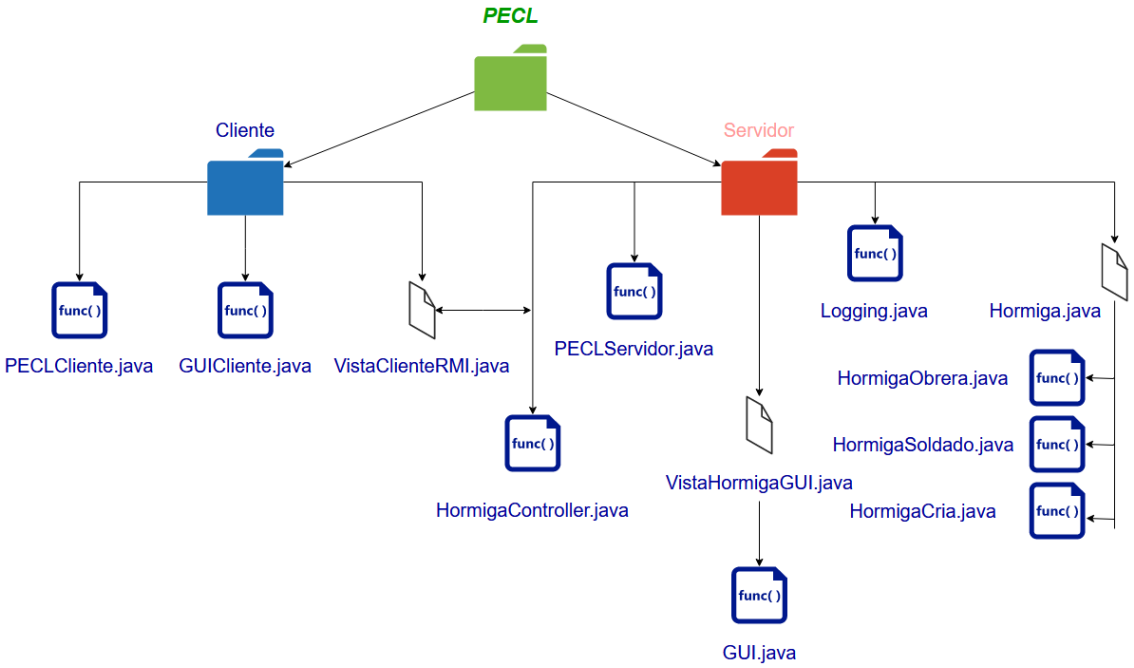
GUI Cliente

Comportamiento similar al GUI del servidor. Se utilizan expresiones lambda, debido a que la programación funcional permite obtener de forma más rápida y eficiente ciertos recursos de la interfaz:

```
protected void crearHiloCambios(GUICliente gui) {
    new Thread(Thread.currentThread().getThreadGroup(),
            new Runnable() {
                public void run() {
                    try {
                        while(true) {
                            try {
                                gui.controlador.cambioAdquireRMI();
                            } catch (InterruptedException ie) {}
                            gui.getTxtfHormigasObrerasExterior().setText(
                                    Long.toString(
                                        gui.controlador.gethFueraColoniaRMI().stream()
                                        .filter(elemento -> elemento.contains("HO"))
                                        .count()
                                    ));
```

Con esto concluye la explicación sobre los atributos y métodos utilizados en las clases del programa, se han omitido los elementos con poca relevancia o ya explicados en apartados anteriores.

Diagrama de clases.



Anexo: Código Fuente.

Proyecto PECLCliente

parte2.GUICliente.java

```
package parte2;


import java.awt.Color;

import java.awt.Font;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JTextField;

import parte2.interfaces.VistaClienteRMI;


import java.awt.event.ActionListener;

import java.rmi.RemoteException;

import java.awt.event.ActionEvent;


public class GUICliente {
```

```
	private VistaClienteRMI controlador;   // Usado para la logica MVC


	private JFrame frmAplicacinHormigas;
	private JButton btnGenerarAmenazaInsecto;
	private JTextField txtfHormigasObrerasExterior;
	private JTextField txtfHormigasObrerasInterior;
	private JTextField txtfHormigasSoldadoInsruccion;
	private JTextField txtfHormigasSoldadoInvasion;
	private JTextField txtfHormigasCriaZonaComer;
	private JTextField txtfHormigasCriaRefugio;

	// Getters and Setters
	public JFrame getFrmAplicacinHormigas() {
		return frmAplicacinHormigas;
	}

	public void setFrmAplicacinHormigas(JFrame frmAplicacinHormigas) {
		this.frmAplicacinHormigas = frmAplicacinHormigas;
	}

	public JTextField getTxtfHormigasObrerasExterior() {
		return txtfHormigasObrerasExterior;
	}

	public void setTxtfHormigasObrerasExterior(JTextField txtfHormigasObrerasExterior) {
		this.txtfHormigasObrerasExterior = txtfHormigasObrerasExterior;
	}

	public JTextField getTxtfHormigasObrerasInterior() {
		return txtfHormigasObrerasInterior;
	}

	public void setTxtfHormigasObrerasInterior(JTextField txtfHormigasObrerasInterior) {
		this.txtfHormigasObrerasInterior = txtfHormigasObrerasInterior;
	}
```

```
public JTextField getTxtfHormigasSoldadoInsruccion() {
	return txtfHormigasSoldadoInsruccion;
}

public void setTxtfHormigasSoldadoInsruccion(JTextField txtfHormigasSoldadoInsruccion) {
	this.txtfHormigasSoldadoInsruccion = txtfHormigasSoldadoInsruccion;
}

public JTextField getTxtfHormigasSoldadoInvasion() {
	return txtfHormigasSoldadoInvasion;
}

public void setTxtfHormigasSoldadoInvasion(JTextField txtfHormigasSoldadoInvasion) {
	this.txtfHormigasSoldadoInvasion = txtfHormigasSoldadoInvasion;
}

public JTextField getTxtfHormigasCriaZonaComer() {
	return txtfHormigasCriaZonaComer;
}

public void setTxtfHormigasCriaZonaComer(JTextField txtfHormigasCriaZonaComer) {
	this.txtfHormigasCriaZonaComer = txtfHormigasCriaZonaComer;
}

public JTextField getTxtfHormigasCriaRefugio() {
	return txtfHormigasCriaRefugio;
}

public void setTxtfHormigasCriaRefugio(JTextField txtfHormigasCriaRefugio) {
	this.txtfHormigasCriaRefugio = txtfHormigasCriaRefugio;
}

public JButton getBtnGenerarAmenazaInsecto() {
	return btnGenerarAmenazaInsecto;
```

```
	}

	public void setBtnGenerarAmenazaInsecto(JButton btnGenerarAmenazaInsecto) {
		this.btnGenerarAmenazaInsecto = btnGenerarAmenazaInsecto;
	}



	/**
	 * Launch the application.
	 */
	/*public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					GUI window = new GUI();
					window.frmAplicacinHormigas.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}*/

	/**
	 * Create the application.
	 */
	public GUICliente(VistaClienteRMI controlador) {
		// Es obligatorio configurar el controlador antes de crear los objetos de la clase
		this.controlador = controlador;
		initialize();
		crearHiloCambios(this);
	}

	/*public GUICliente() {
		initialize();
```

```
}*/

protected void crearHiloCambios(GUICliente gui) {
        new Thread(Thread.currentThread().getThreadGroup(),
                new Runnable() {
                        public void run() {
                                try {
                                        while(true) {
                                                try {

gui.controlador.cambioAdquireRMI();
                                                } catch (InterruptedException ie) {}

gui.getTxtfHormigasObrerasExterior().setText(
                                                        Long.toString(

gui.controlador.gethFueraColoniaRMI().stream()

.filter(elemento -> elemento.contains("HO"))
                                                                .count()
                                                        ));

gui.getTxtfHormigasObrerasInterior().setText(
                                                        Long.toString(

gui.controlador.gethDentroColoniaRMI().stream()

.filter(elemento -> elemento.contains("HO"))
                                                                .count()
                                                        ));

gui.getTxtfHormigasSoldadoInsruccion().setText(
                                                        Long.toString(

gui.controlador.gethZonaInstruccionRMI().stream()
                                                                .count()
                                                        ));
```

```
                gui.getTxtfHormigasSoldadoInvasion().setText(
                                                                Long.toString(
                gui.controlador.gethRepeliendoInsectoRMI().stream()
                                                                        .count()
                                                                ));
                gui.getTxtfHormigasCriaZonaComer().setText(
                                                                Long.toString(
                gui.controlador.gethZonaComerRMI().stream()
                .filter(elemento -> elemento.contains("HC"))
                                                                        .count()
                                                                ));
                gui.getTxtfHormigasCriaRefugio().setText(
                                                                Long.toString(
                gui.controlador.gethRefugioRMI().stream()
                                                                        .count()
                                                                ));
                                                        }
                                                } catch (Exception e) {
                                                        e.printStackTrace();
                                                }
                                        }
                                },
                                "HiloEventoControlador"
                        ).start();
        }


        /**
         * Initialize the contents of the frame.
```

```
		*/
		private void initialize() {
			frmAplicacinHormigas = new JFrame();
			frmAplicacinHormigas.setResizable(false);
			frmAplicacinHormigas.setTitle("Aplicación Hormigas Cliente");
			frmAplicacinHormigas.setBounds(100, 100, 715, 494);
			frmAplicacinHormigas.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			frmAplicacinHormigas.getContentPane().setLayout(null);

			JLabel lblHormigasObrerasExterior = new JLabel("Número de hormigas obreras en el exterior de la colonia:");
			lblHormigasObrerasExterior.setFont(new Font("Tahoma", Font.PLAIN, 14));
			lblHormigasObrerasExterior.setBounds(126, 79, 353, 28);
			frmAplicacinHormigas.getContentPane().add(lblHormigasObrerasExterior);

			JLabel lblHormigasObrerasInterior = new JLabel("Número de hormigas obreras en el interior de la colonia:");
			lblHormigasObrerasInterior.setFont(new Font("Tahoma", Font.PLAIN, 14));
			lblHormigasObrerasInterior.setBounds(126, 132, 353, 14);
			frmAplicacinHormigas.getContentPane().add(lblHormigasObrerasInterior);

			btnGenerarAmenazaInsecto = new JButton("Generar Amenaza Insecto Invasor");
			btnGenerarAmenazaInsecto.setFont(new Font("Tahoma", Font.PLAIN, 18));
			btnGenerarAmenazaInsecto.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					try {
						controlador.generarInsectoInvasorRMI();
					} catch (RemoteException e1) {
						e1.printStackTrace();
					}
				}
			});
			btnGenerarAmenazaInsecto.setBounds(170, 349, 389, 41);
			frmAplicacinHormigas.getContentPane().add(btnGenerarAmenazaInsecto);
```

```
txtfHormigasObrerasExterior = new JTextField();
txtfHormigasObrerasExterior.setForeground(Color.BLACK);
txtfHormigasObrerasExterior.setEditable(false);
txtfHormigasObrerasExterior.setColumns(10);
txtfHormigasObrerasExterior.setBounds(520, 82, 89, 26);
frmAplicacinHormigas.getContentPane().add(txtfHormigasObrerasExterior);

txtfHormigasObrerasInterior = new JTextField();
txtfHormigasObrerasInterior.setForeground(Color.BLACK);
txtfHormigasObrerasInterior.setEditable(false);
txtfHormigasObrerasInterior.setColumns(10);
txtfHormigasObrerasInterior.setBounds(520, 128, 89, 26);
frmAplicacinHormigas.getContentPane().add(txtfHormigasObrerasInterior);

JLabel lblHormigasSoldadoInstruccion = new JLabel("Número de hormigas soldado haciendo
instrucción:");
lblHormigasSoldadoInstruccion.setFont(new Font("Tahoma", Font.PLAIN, 14));
lblHormigasSoldadoInstruccion.setBounds(126, 165, 353, 28);
frmAplicacinHormigas.getContentPane().add(lblHormigasSoldadoInstruccion);

txtfHormigasSoldadoInsruccion = new JTextField();
txtfHormigasSoldadoInsruccion.setForeground(Color.BLACK);
txtfHormigasSoldadoInsruccion.setEditable(false);
txtfHormigasSoldadoInsruccion.setColumns(10);
txtfHormigasSoldadoInsruccion.setBounds(520, 168, 89, 26);
frmAplicacinHormigas.getContentPane().add(txtfHormigasSoldadoInsruccion);

JLabel lblHormigasSoldadoInvasion = new JLabel("Número de hormigas soldado repeliendo
una invasión:");
lblHormigasSoldadoInvasion.setFont(new Font("Tahoma", Font.PLAIN, 14));
lblHormigasSoldadoInvasion.setBounds(126, 204, 353, 28);
frmAplicacinHormigas.getContentPane().add(lblHormigasSoldadoInvasion);

txtfHormigasSoldadoInvasion = new JTextField();
txtfHormigasSoldadoInvasion.setForeground(Color.BLACK);
```

```
                txtfHormigasSoldadoInvasion.setEditable(false);
                txtfHormigasSoldadoInvasion.setColumns(10);
                txtfHormigasSoldadoInvasion.setBounds(520, 207, 89, 26);
                frmAplicacinHormigas.getContentPane().add(txtfHormigasSoldadoInvasion);

                JLabel lblHormigasCriaZonaComer = new JLabel("Número de hormigas crías en la ZONA PARA
COMER:");
                lblHormigasCriaZonaComer.setFont(new Font("Tahoma", Font.PLAIN, 14));
                lblHormigasCriaZonaComer.setBounds(126, 243, 353, 28);
                frmAplicacinHormigas.getContentPane().add(lblHormigasCriaZonaComer);

                txtfHormigasCriaZonaComer = new JTextField();
                txtfHormigasCriaZonaComer.setForeground(Color.BLACK);
                txtfHormigasCriaZonaComer.setEditable(false);
                txtfHormigasCriaZonaComer.setColumns(10);
                txtfHormigasCriaZonaComer.setBounds(520, 246, 89, 26);
                frmAplicacinHormigas.getContentPane().add(txtfHormigasCriaZonaComer);

                JLabel lblHormigasCriaRefugio = new JLabel("Número de hormigas crías en el REFUGIO:");
                lblHormigasCriaRefugio.setFont(new Font("Tahoma", Font.PLAIN, 14));
                lblHormigasCriaRefugio.setBounds(126, 282, 353, 28);
                frmAplicacinHormigas.getContentPane().add(lblHormigasCriaRefugio);

                txtfHormigasCriaRefugio = new JTextField();
                txtfHormigasCriaRefugio.setForeground(Color.BLACK);
                txtfHormigasCriaRefugio.setEditable(false);
                txtfHormigasCriaRefugio.setColumns(10);
                txtfHormigasCriaRefugio.setBounds(520, 285, 89, 26);
                frmAplicacinHormigas.getContentPane().add(txtfHormigasCriaRefugio);
        }
}
```

PECLCliente.java

```
package emilio.peclcliente;

import java.awt.EventQueue;
```

```
import java.rmi.Naming;
import parte2.interfaces.VistaClienteRMI;

import java.awt.EventQueue;
import java.rmi.Naming;
import parte2.GUICliente;
import parte2.interfaces.VistaClienteRMI;

public class PECLCliente {

	public static void main(String[] args) {

		//	Inicializo el controlador
		try {
			VistaClienteRMI controlador = (VistaClienteRMI)
Naming.lookup("//127.0.0.1/ControladorHormiga");

			/**
			 * Launch the GUI application.
			 */

			EventQueue.invokeLater(new Runnable() {
				public void run() {
					try {
						GUICliente window = new GUICliente(controlador);
						window.getFrmAplicacinHormigas().setVisible(true);
					} catch (Exception e) {
						e.printStackTrace();
					}
				}
			});

		} catch (Exception e) {e.printStackTrace();}
```

```
		}


}
```

parte2.interfaces.VistaClienteRMI.java

```
package parte2.interfaces;


import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.Set;


/**

 * @author Emilio Macías Do Santos

 *

 *		Esta interfaz sirve para que la vista GUI.java pueda saber qué métodos llamar al controlador,

 *		de esta forma se implementa el MVC.

 *

 *		La vista no verá nada más que los métodos que deba saber.

 */

public interface VistaClienteRMI extends Remote{

		// Getter y Setters públicos

		public Set<String> gethDentroColoniaRMI() throws RemoteException;


		public Set<String> gethFueraColoniaRMI() throws RemoteException;


		public Set<String> gethZonaInstruccionRMI() throws RemoteException;


		public Set<String> gethRepeliendoInsectoRMI() throws RemoteException;


		public Set<String> gethZonaComerRMI() throws RemoteException;


		public Set<String> gethRefugioRMI() throws RemoteException;


		// Método para generar un insecto invasor

		public void generarInsectoInvasorRMI() throws RemoteException;
```

```
        // Métodos que utilizan un semáforo para controlar los eventos.
        public void cambioAdquireRMI() throws InterruptedException, RemoteException;

        public void cambioReleaseRMI() throws RemoteException;

}
```

Proyecto PECLServidor

parte1.GUI.java

```
package parte1;

import java.awt.Color;
import java.awt.Font;
import java.awt.TextArea;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import parte1.interfaces.VistaHormigaGUI;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class GUI {

        private VistaHormigaGUI controlador;        // Usado para la logica MVC

        private JFrame frmAplicacinHormigas;
        private JTextField txtfHormigasBuscandoComida;
        private JTextField txtfHormigasAlmacenComida;
        private JTextField txtfHomigasLlevandoZonaComer;
        private JTextField txtfHomigasInstruccion;
```

```
private JTextField txtfHomigasDescansando;
private JTextField txtfUnidadesAlmacen;
private JTextField txtfUnidadesZonaComer;
// Adaptado
private TextArea txtaHormigasRepeliendoInsecto;
private TextArea txtaZonaParaComer;
private TextArea txtaRefugio;
private JButton btnGenerarAmenazaInsecto;

// Getters and Setters
public JFrame getFrmAplicacinHormigas() {
	return frmAplicacinHormigas;
}

public void setFrmAplicacinHormigas(JFrame frmAplicacinHormigas) {
	this.frmAplicacinHormigas = frmAplicacinHormigas;
}

public JTextField getTxtfHormigasBuscandoComida() {
	return txtfHormigasBuscandoComida;
}

public void setTxtfHormigasBuscandoComida(JTextField txtfHormigasBuscandoComida) {
	this.txtfHormigasBuscandoComida = txtfHormigasBuscandoComida;
}

public JTextField getTxtfHormigasAlmacenComida() {
	return txtfHormigasAlmacenComida;
}

public void setTxtfHormigasAlmacenComida(JTextField txtfHormigasAlmacenComida) {
	this.txtfHormigasAlmacenComida = txtfHormigasAlmacenComida;
}

public JTextField getTxtfHomigasLlevandoZonaComer() {
```

```
		return txtfHomigasLlevandoZonaComer;
	}

	public void setTxtfHomigasLlevandoZonaComer(JTextField txtfHomigasLlevandoZonaComer) {
		this.txtfHomigasLlevandoZonaComer = txtfHomigasLlevandoZonaComer;
	}

	public JTextField getTxtfHomigasInstruccion() {
		return txtfHomigasInstruccion;
	}

	public void setTxtfHomigasInstruccion(JTextField txtfHomigasInstruccion) {
		this.txtfHomigasInstruccion = txtfHomigasInstruccion;
	}

	public JTextField getTxtfHomigasDescansando() {
		return txtfHomigasDescansando;
	}

	public void setTxtfHomigasDescansando(JTextField txtfHomigasDescansando) {
		this.txtfHomigasDescansando = txtfHomigasDescansando;
	}

	public JTextField getTxtfUnidadesAlmacen() {
		return txtfUnidadesAlmacen;
	}

	public void setTxtfUnidadesAlmacen(JTextField txtfUnidadesAlmacen) {
		this.txtfUnidadesAlmacen = txtfUnidadesAlmacen;
	}

	public JTextField getTxtfUnidadesZonaComer() {
		return txtfUnidadesZonaComer;
	}
```

```
public void setTxtfUnidadesZonaComer(JTextField txtfUnidadesZonaComer) {
	this.txtfUnidadesZonaComer = txtfUnidadesZonaComer;
}

public TextArea getTxtaHormigasRepeliendoInsecto() {
	return txtaHormigasRepeliendoInsecto;
}

public void setTxtaHormigasRepeliendoInsecto(TextArea txtaHormigasRepeliendoInsecto) {
	this.txtaHormigasRepeliendoInsecto = txtaHormigasRepeliendoInsecto;
}

public TextArea getTxtaZonaParaComer() {
	return txtaZonaParaComer;
}

public void setTxtaZonaParaComer(TextArea txtaZonaParaComer) {
	this.txtaZonaParaComer = txtaZonaParaComer;
}

public TextArea getTxtaRefugio() {
	return txtaRefugio;
}

public void setTxtaRefugio(TextArea txtaRefugio) {
	this.txtaRefugio = txtaRefugio;
}

public JButton getBtnGenerarAmenazaInsecto() {
	return btnGenerarAmenazaInsecto;
}

public void setBtnGenerarAmenazaInsecto(JButton btnGenerarAmenazaInsecto) {
	this.btnGenerarAmenazaInsecto = btnGenerarAmenazaInsecto;
}
```

```
/**
 * Launch the application.
 */
/*public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				GUI window = new GUI();
				window.frmAplicacinHormigas.setVisible(true);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
}*/

/**
 * Create the application.
 */
public GUI(VistaHormigaGUI controlador) {
	// Es obligatorio configurar el controlador antes de crear los objetos de la clase
	this.controlador = controlador;
	initialize();
	crearHiloCambios(this);
}

protected void crearHiloCambios(GUI gui) {
	new Thread(Thread.currentThread().getThreadGroup(),
			new Runnable() {
				public void run() {
					try {
						while(true) {
							try {
```

```
            gui.controlador.cambioAdquire();
                                                                    } catch (InterruptedException ie) {}

            gui.getTxtfUnidadesAlmacen().setText(Integer.toString(gui.controlador.getUnidadesComidaAlmacen()));

            gui.getTxtfUnidadesZonaComer().setText(gui.controlador.getUnidadesComidaZonaComer().toString());

            gui.getTxtfHormigasBuscandoComida().setText(String.join(",", gui.controlador.gethBuscandoComida()));

            gui.getTxtaHormigasRepeliendoInsecto().setText(String.join(",",
gui.controlador.gethRepeliendoInsecto()));

            gui.getTxtfHormigasAlmacenComida().setText(String.join(",", gui.controlador.gethAlmacenComida()));

            gui.getTxtfHomigasLlevandoZonaComer().setText(String.join(",",
gui.controlador.gethLlevandoZonaComer()));

            gui.getTxtfHomigasInstruccion().setText(String.join(",", gui.controlador.gethZonaInstruccion()));

            gui.getTxtfHomigasDescansando().setText(String.join(",", gui.controlador.gethZonaDescanso()));

            gui.getTxtaZonaParaComer().setText(String.join(",", gui.controlador.gethZonaComer()));

            gui.getTxtaRefugio().setText(String.join(",", gui.controlador.gethRefugio()));
                                                                    }
                                                            } catch (Exception e) {
                                                                    e.printStackTrace();
                                                            }
                                                    }
                                            },
                                            "HiloEventoControlador"
                                    ).start();
            }

            /**
             * Initialize the contents of the frame.
             */
```

```
	private void initialize() {
		frmAplicacinHormigas = new JFrame();
		frmAplicacinHormigas.setResizable(false);
		frmAplicacinHormigas.setTitle("Aplicación Hormigas Servidor");
		frmAplicacinHormigas.setBounds(100, 100, 941, 705);
		frmAplicacinHormigas.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frmAplicacinHormigas.getContentPane().setLayout(null);

		JLabel lblExteriordelaColonia = new JLabel("Exterior de la colonia:");
		lblExteriordelaColonia.setBounds(7, 7, 251, 17);
		lblExteriordelaColonia.setFont(new Font("Tahoma", Font.BOLD, 14));
		frmAplicacinHormigas.getContentPane().add(lblExteriordelaColonia);

		JLabel lblHormigasBuscandoComida = new JLabel("Hormigas buscando comida:\r\n");
		lblHormigasBuscandoComida.setBounds(7, 31, 309, 14);
		lblHormigasBuscandoComida.setLabelFor(lblExteriordelaColonia);
		frmAplicacinHormigas.getContentPane().add(lblHormigasBuscandoComida);

		txtfHormigasBuscandoComida = new JTextField();
		txtfHormigasBuscandoComida.setForeground(new Color(0, 0, 0));
		txtfHormigasBuscandoComida.setEditable(false);
		txtfHormigasBuscandoComida.setBounds(326, 31, 503, 20);
		frmAplicacinHormigas.getContentPane().add(txtfHormigasBuscandoComida);
		txtfHormigasBuscandoComida.setColumns(10);

		JLabel lblHormigasRepeliendoInsecto = new JLabel("Hormigas repeliendo un insecto
invasor:");
		lblHormigasRepeliendoInsecto.setBounds(7, 84, 313, 14);
		frmAplicacinHormigas.getContentPane().add(lblHormigasRepeliendoInsecto);

		txtaHormigasRepeliendoInsecto = new TextArea();
		txtaHormigasRepeliendoInsecto.setEditable(false);
		txtaHormigasRepeliendoInsecto.setForeground(new Color(0, 0, 0));
		txtaHormigasRepeliendoInsecto.setBounds(326, 87, 503, 160);
```

```
frmAplicacinHormigas.getContentPane().add(txtaHormigasRepeliendoInsecto);

JLabel lblInteriordelaColonia = new JLabel("Interior de la colonia:");
lblInteriordelaColonia.setBounds(7, 245, 251, 17);
lblInteriordelaColonia.setFont(new Font("Tahoma", Font.BOLD, 14));
frmAplicacinHormigas.getContentPane().add(lblInteriordelaColonia);

JLabel lblHormigasAlmacenComida = new JLabel("Hormigas en el ALMACÉN DE COMIDA:");
lblHormigasAlmacenComida.setBounds(7, 269, 313, 14);
frmAplicacinHormigas.getContentPane().add(lblHormigasAlmacenComida);

txtfHormigasAlmacenComida = new JTextField();
txtfHormigasAlmacenComida.setEditable(false);
txtfHormigasAlmacenComida.setBounds(326, 269, 503, 20);
frmAplicacinHormigas.getContentPane().add(txtfHormigasAlmacenComida);
txtfHormigasAlmacenComida.setColumns(10);

JLabel lblHomigasLlevandoZonaComer = new JLabel("Hormigas llevando comida a la ZONA
PARA COMER:");
lblHomigasLlevandoZonaComer.setBounds(7, 293, 313, 14);
frmAplicacinHormigas.getContentPane().add(lblHomigasLlevandoZonaComer);

txtfHomigasLlevandoZonaComer = new JTextField();
txtfHomigasLlevandoZonaComer.setEditable(false);
txtfHomigasLlevandoZonaComer.setBounds(326, 293, 503, 20);
frmAplicacinHormigas.getContentPane().add(txtfHomigasLlevandoZonaComer);
txtfHomigasLlevandoZonaComer.setColumns(10);

JLabel lblHomigasInstruccion = new JLabel("Hormigas haciendo INSTRUCCIÓN");
lblHomigasInstruccion.setBounds(7, 317, 309, 14);
frmAplicacinHormigas.getContentPane().add(lblHomigasInstruccion);

txtfHomigasInstruccion = new JTextField();
txtfHomigasInstruccion.setEditable(false);
```

```
txtfHomigasInstruccion.setBounds(326, 317, 503, 17);
frmAplicacinHormigas.getContentPane().add(txtfHomigasInstruccion);
txtfHomigasInstruccion.setColumns(10);

JLabel lblHormigasDescansando = new JLabel("Hormigas descansando:");
lblHormigasDescansando.setBounds(7, 338, 309, 14);
frmAplicacinHormigas.getContentPane().add(lblHormigasDescansando);

txtfHomigasDescansando = new JTextField();
txtfHomigasDescansando.setEditable(false);
txtfHomigasDescansando.setBounds(326, 338, 236, 20);
frmAplicacinHormigas.getContentPane().add(txtfHomigasDescansando);
txtfHomigasDescansando.setColumns(10);

JLabel lblUnidadesAlmacen = new JLabel("Unidades de Comida (ALMACÉN)");
lblUnidadesAlmacen.setHorizontalAlignment(SwingConstants.RIGHT);
lblUnidadesAlmacen.setBounds(572, 341, 267, 14);
frmAplicacinHormigas.getContentPane().add(lblUnidadesAlmacen);

txtfUnidadesAlmacen = new JTextField();
txtfUnidadesAlmacen.setEditable(false);
txtfUnidadesAlmacen.setBounds(849, 338, 42, 20);
frmAplicacinHormigas.getContentPane().add(txtfUnidadesAlmacen);
txtfUnidadesAlmacen.setColumns(10);

JLabel lblUnidadesZonaComer = new JLabel("Unidades de Comida (ZONA PARA COMER)");
lblUnidadesZonaComer.setHorizontalAlignment(SwingConstants.RIGHT);
lblUnidadesZonaComer.setBounds(572, 373, 272, 14);
frmAplicacinHormigas.getContentPane().add(lblUnidadesZonaComer);

txtfUnidadesZonaComer = new JTextField();
txtfUnidadesZonaComer.setEditable(false);
txtfUnidadesZonaComer.setColumns(10);
txtfUnidadesZonaComer.setBounds(849, 370, 42, 20);
```

```
frmAplicacinHormigas.getContentPane().add(txtfUnidadesZonaComer);

JLabel lblZonaParaComer = new JLabel("ZONA PARA COMER:");
lblZonaParaComer.setBounds(7, 419, 313, 14);
frmAplicacinHormigas.getContentPane().add(lblZonaParaComer);

txtaZonaParaComer = new TextArea();
txtaZonaParaComer.setEditable(false);
txtaZonaParaComer.setBounds(326, 422, 503, 84);
frmAplicacinHormigas.getContentPane().add(txtaZonaParaComer);

JLabel lblRefugio = new JLabel("REFUGIO:");
lblRefugio.setBounds(7, 509, 309, 14);
frmAplicacinHormigas.getContentPane().add(lblRefugio);

txtaRefugio = new TextArea();
txtaRefugio.setEditable(false);
txtaRefugio.setBounds(326, 512, 503, 84);
frmAplicacinHormigas.getContentPane().add(txtaRefugio);

JButton btnPausar = new JButton("Pausar");
btnPausar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                controlador.pararEjecucion();
                btnGenerarAmenazaInsecto.setEnabled(false);
        }
});
btnPausar.setBounds(32, 618, 89, 23);
frmAplicacinHormigas.getContentPane().add(btnPausar);

JButton btnReanudar = new JButton("Reanudar");
btnReanudar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                controlador.continuarEjecucion();
```

```
                                btnGenerarAmenazaInsecto.setEnabled(true);
                        }
                });
                btnReanudar.setBounds(169, 618, 89, 23);
                frmAplicacinHormigas.getContentPane().add(btnReanudar);


                btnGenerarAmenazaInsecto = new JButton("Generar Amenaza Insecto Invasor");
                btnGenerarAmenazaInsecto.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                controlador.generarInsectoInvasor();
                        }
                });
                btnGenerarAmenazaInsecto.setBounds(572, 618, 257, 23);
                frmAplicacinHormigas.getContentPane().add(btnGenerarAmenazaInsecto);
        }
}
```

parte1.HormigaController.java

```
package parte1;


import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Set;
import java.util.concurrent.CopyOnWriteArraySet;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.Semaphore;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;


import parte1.Logging.LoggingException;
import parte1.interfaces.Hormiga;
import parte1.interfaces.VistaHormigaGUI;
import parte2.interfaces.VistaClienteRMI;


public class HormigaController extends UnicastRemoteObject implements Runnable, VistaHormigaGUI,
VistaClienteRMI {
```

```
/**
 *
 */
private static final long serialVersionUID = 7811150400089455866L;
// Semáforos para los tuneles
private Semaphore semTunelEntrada;
private Semaphore semTunelSalida;
// Semáforos para las zonas compartidas;
private Semaphore semAlmacenComida;
private Semaphore semUnidadesAlmacenVacio;
private Semaphore semUnidadesAlmacenExclusion;
private Semaphore semUnidadesAlmacenLleno;
private Semaphore semUnidadesZonaComer;
// Semáforo para notificar a las hormigas cria
private Semaphore semLuchandoInsecto;
// Atributos
private Logging log;
// Semáforo para notificar el cambio en alguno de los recursos compartidos
private Semaphore semCambioRecursos;
// Recursos compartidos para saber donde estás las hormigas en cada zona
private Set<String> hDentroColonia;
private Set<String> hFueraColonia;
private Set<String> hBuscandoComida;
private Set<String> hRepeliendoInsecto;
private Set<String> hAlmacenComida;
private Set<String> hLlevandoZonaComer;
private Set<String> hZonaInstruccion;
private Set<String> hZonaDescanso;
private Set<String> hZonaComer;
private Set<String> hRefugio;
// Recursos compartidos para saber las unidades de comida
private int unidadesComidaAlmacen;
private AtomicInteger unidadesComidaZonaComer;
// Atributo para poder parar la ejecución del hormiguero
private AtomicBoolean pararEjecucion;
```

```
	// Atributos estáticos para obtener los grupos de hilos de las hormigas
	private static ThreadGroup hormigasObreras;
	private static ThreadGroup hormigasSoldado;
	private static ThreadGroup hormigasCria;
	// Atributo estático para generar un insecto invasor
	private static AtomicBoolean luchandoInsectoInvasor = new AtomicBoolean(false);

	public HormigaController() throws RemoteException{
		// Me creo la cola del Logging
		this.log = new Logging(new LinkedBlockingQueue<String>());
		// Semáforos para los túneles
		this.semTunelEntrada = new Semaphore(1,true);
		this.semTunelSalida = new Semaphore(2,true);
		// Semáforo para el cambio de los recursos compartidos
		this.semCambioRecursos = new Semaphore(0,true);
		// Semáforos para las zonas compartidas
		this.semAlmacenComida = new Semaphore(10,true);
		this.semUnidadesAlmacenVacio = new Semaphore(0,true);
		this.semUnidadesAlmacenExclusion = new Semaphore(1,true);
		this.semUnidadesAlmacenLleno = new
Semaphore(this.semAlmacenComida.availablePermits(),true);
		this.semUnidadesZonaComer= new Semaphore(0,true);
		// Listas de Zonas
		this.hDentroColonia = new CopyOnWriteArraySet<String>();
		this.hFueraColonia = new CopyOnWriteArraySet<String>();
		this.hBuscandoComida = new CopyOnWriteArraySet<String>();
		this.hRepeliendoInsecto = new CopyOnWriteArraySet<String>();
		this.hAlmacenComida = new CopyOnWriteArraySet<String>();
		this.hLlevandoZonaComer = new CopyOnWriteArraySet<String>();
		this.hZonaInstruccion = new CopyOnWriteArraySet<String>();
		this.hZonaDescanso = new CopyOnWriteArraySet<String>();
		this.hZonaComer = new CopyOnWriteArraySet<String>();
		this.hRefugio = new CopyOnWriteArraySet<String>();
		// Unidades de comida común
		this.unidadesComidaAlmacen = 0;
```

```
		this.unidadesComidaZonaComer = new AtomicInteger(0);
		// Atributo para parar la ejecucion
		this.pararEjecucion = new AtomicBoolean(false);
		this.semLuchandoInsecto = new Semaphore(10000,true);
	}

	// Getter y Setters públicos
	public Set<String> gethDentroColonia() {
		return hDentroColonia;
	}

	public Set<String> gethFueraColonia() {
		return hFueraColonia;
	}

	public Set<String> gethBuscandoComida() {
		return hBuscandoComida;
	}

	public Set<String> gethRepeliendoInsecto() {
		return hRepeliendoInsecto;
	}

	public Set<String> gethAlmacenComida() {
		return hAlmacenComida;
	}

	public Set<String> gethLlevandoZonaComer() {
		return hLlevandoZonaComer;
	}

	public Set<String> gethZonaInstruccion() {
		return hZonaInstruccion;
	}
```

```
public Set<String> gethZonaDescanso() {
	return hZonaDescanso;
}

public Set<String> gethZonaComer() {
	return hZonaComer;
}

public Set<String> gethRefugio() {
	return hRefugio;
}

public int getUnidadesComidaAlmacen() {
	return unidadesComidaAlmacen;
}

public void setUnidadesComidaAlmacen(int unidadesComidaAlmacen) {
	this.unidadesComidaAlmacen = unidadesComidaAlmacen;
}

public AtomicInteger getUnidadesComidaZonaComer() {
	return unidadesComidaZonaComer;
}

public void setUnidadesComidaZonaComer(AtomicInteger unidadesComidaZonaComer) {
	this.unidadesComidaZonaComer = unidadesComidaZonaComer;
}
public static ThreadGroup getHormigasObreras() {
	return hormigasObreras;
}

public static void setHormigasObreras(ThreadGroup hormigasObreras) {
	HormigaController.hormigasObreras = hormigasObreras;
}
```

```
public static ThreadGroup getHormigasSoldado() {
	return hormigasSoldado;
}

public static void setHormigasSoldado(ThreadGroup hormigasSoldado) {
	HormigaController.hormigasSoldado = hormigasSoldado;
}

public static ThreadGroup getHormigasCria() {
	return hormigasCria;
}

public static void setHormigasCria(ThreadGroup hormigasCria) {
	HormigaController.hormigasCria = hormigasCria;
}

public static AtomicBoolean getLuchandoInsectoInvasor() {
	return luchandoInsectoInvasor;
}

public Semaphore getSemLuchandoInsecto() {
	return semLuchandoInsecto;
}

@Override
public AtomicBoolean getPararEjecucion() {
	return pararEjecucion;
}
@Override
public void pararEjecucion() {
	this.pararEjecucion.compareAndSet(false, true);
}
@Override
public void continuarEjecucion() {
	if(this.pararEjecucion.getAndSet(false) == true) { // Si el valor anterior era true
```

```
                        try {
                                hormigasObreras.resume();
                                hormigasSoldado.resume();
                                hormigasCria.resume();
                        } catch(IllegalMonitorStateException e) {}


                }
        }

        @Override
        public void generarInsectoInvasor() {
                if(luchandoInsectoInvasor.getAndSet(true) == false) {
                        if(this.pararEjecucion.get()) {
                                luchandoInsectoInvasor.set(false);
                                return;
                        }
                        this.semLuchandoInsecto.drainPermits();
                        hormigasSoldado.interrupt();
                        hormigasCria.interrupt();
                        new Thread(Thread.currentThread().getThreadGroup(),
                                new Runnable() {
                                        @Override
                                        public void run() {
                                                try {
                                                        Thread.sleep(20*1000);
                                                } catch (InterruptedException e) {}
                                                finally {
                                                        luchandoInsectoInvasor.set(false);
                                                        semLuchandoInsecto.release(10000);
                                                }
                                        }
                        }, "HiloControlInsectoInvasor").start();;
                }
        }
```

```
// Métodos sobreescritos de la interfaz VistaClienteRMI
@Override
public Set<String> gethDentroColoniaRMI() throws RemoteException {
	return this.gethDentroColonia();
}

@Override
public Set<String> gethFueraColoniaRMI() throws RemoteException {
	return this.gethFueraColonia();
}

@Override
public Set<String> gethZonaInstruccionRMI() throws RemoteException {
	return this.gethZonaInstruccion();
}

@Override
public Set<String> gethRepeliendoInsectoRMI() throws RemoteException {
	return this.gethRepeliendoInsecto();
}

@Override
public Set<String> gethZonaComerRMI() throws RemoteException {
	return this.gethZonaComer();
}

@Override
public Set<String> gethRefugioRMI() throws RemoteException {
	return this.gethRefugio();
}

@Override
public void generarInsectoInvasorRMI() throws RemoteException {
	this.generarInsectoInvasor();
```

```
	}

	@Override
	public void cambioAdquireRMI() throws InterruptedException, RemoteException {
		this.cambioAdquire();
	}

	@Override
	public void cambioReleaseRMI() throws RemoteException {
		this.cambioRelease();

	}

	@Override
	public void cambioAdquire() throws InterruptedException {
		this.semCambioRecursos.drainPermits();
		this.semCambioRecursos.acquire();
	}
	@Override
	public void cambioRelease() {
		this.semCambioRecursos.release();
	}

	// Métodos
	public void addLog(Hormiga hormiga, String mensaje) throws InterruptedException {
		log.getListaLog().put(Logging.getMensaje("La hormiga "+ hormiga.getNombreHormiga() +
mensaje));
	}

	//	Métodos para los túneles
	public void pasarTunelEntrada(Hormiga hormiga) {
		try {
			this.semTunelEntrada.acquire();
			this.hFueraColonia.remove(hormiga.getNombreHormiga());
			this.cambioRelease();
```

```
                        // Está la hormiga en el tunel
                        this.addLog(hormiga, '', ha pasado por un tunel de entrada.");
                        Thread.sleep(100);
                        this.addLog(hormiga, '', ha salido por un tunel de entrada.");
                } catch(InterruptedException ie) {}
                finally {
                        this.semTunelEntrada.release();
                        this.hDentroColonia.add(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }

        public void pasarTunelSalida(Hormiga hormiga) {
                try {
                        this.semTunelSalida.acquire();
                        this.hDentroColonia.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                        // Está la hormiga en el tunel
                        this.addLog(hormiga, '', ha pasado por un tunel de salida.");
                        Thread.sleep(100);
                        this.addLog(hormiga, '', ha salido por un tunel de salida.");
                } catch(InterruptedException ie) {}
                finally {
                        this.semTunelSalida.release();
                        this.hFueraColonia.add(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }

        // Métodos para controlar el acceso a las zonas compartidas.

        // Métodos hormigas obreras en zonas compartidas.
        public void transicionBuscandoComida(HormigaObrera hormiga) {
                try {
```

```
                        this.addLog(hormiga, ", está buscando comida en el campo.");
                        this.hBuscandoComida.add(hormiga.getName()); // Queda reflejado que está dentro
del almacen
                        this.cambioRelease();
                        Thread.sleep(4*1000); // Tiempo que tarda en depositarlo
                        this.addLog(hormiga, ", ha terminado de buscar comida y se prepara para entrar.");
                } catch(InterruptedException ie) {}
                finally {
                        this.hBuscandoComida.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }

        public void entrarAlmacenComidaDepositar(HormigaObrera hormiga) {
                try {
                        this.semUnidadesAlmacenLleno.acquire();
                        this.semAlmacenComida.acquire(); // Entra en el almacen de comida
                        // Está la hormiga en el tunel
                        this.addLog(hormiga, ", ha entrado al almacén de comida a depositar.");
                        this.hAlmacenComida.add(hormiga.getName()); // Queda reflejado que está dentro
del almacen
                        this.cambioRelease();
                        Thread.sleep((long) (((Math.random()*2)+2)*1000)); // Tiempo que tarda en
depositarlo
                        this.semUnidadesAlmacenExclusion.acquire(); // Exclusion
                        this.unidadesComidaAlmacen+=5; // Deposita la comida en el almacén
                        this.semUnidadesAlmacenExclusion.release(); // Exclusion
                        this.cambioRelease();
                        this.addLog(hormiga, ", ha depositado la comida y ha salido del almacén.");
                } catch(InterruptedException ie) {}
                finally {
                        this.semAlmacenComida.release();
                        this.semUnidadesAlmacenVacio.release(); // Sincronizacion
                        this.hAlmacenComida.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
```

```
	}


	public void entrarAlmacenComidaConsumir(HormigaObrera hormiga) {
		try {
			this.semUnidadesAlmacenVacio.acquire(); // Sincronización
			this.semAlmacenComida.acquire(); // Entra en el almacen de comida
			// Está la hormiga en el tunel
			this.addLog(hormiga, '', ha entrado al almacén de comida a consumir.");
			this.hAlmacenComida.add(hormiga.getName()); // Queda reflejado que está dentro del almacen
			this.cambioRelease();
			Thread.sleep((long) (((Math.random()*1)+1)*1000)); // Tiempo que tarda en depositarlo
			this.semUnidadesAlmacenExclusion.acquire(); // Exclusion
			this.unidadesComidaAlmacen-=5;
			this.semUnidadesAlmacenExclusion.release();
			this.cambioRelease();
			this.addLog(hormiga, '', ha recogido la comida y ha salido del almacén.");
		} catch(InterruptedException ie) {}
		finally {
			this.semAlmacenComida.release();
			this.semUnidadesAlmacenLleno.release();
			this.hAlmacenComida.remove(hormiga.getNombreHormiga());
			this.cambioRelease();
		}
	}


	public void transicionZonaComer(HormigaObrera hormiga) {
		try {
			this.addLog(hormiga, '', está haciendo el camino para la zona para comer.");
			this.hLlevandoZonaComer.add(hormiga.getName()); // Queda reflejado que está dentro del almacen
			this.cambioRelease();
			Thread.sleep((long) (((Math.random()*2)+1)*1000)); // Tiempo que tarda en depositarlo
			this.addLog(hormiga, '', ha llegado a la zona comer.");
```

```
                } catch(InterruptedException ie) {}
                finally {
                        this.hLlevandoZonaComer.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }

        public void entrarZonaComerDepositar(HormigaObrera hormiga, int unidadesADepositar) {
                try {
                        this.addLog(hormiga, '', ha entrado a la zona comer a depositar.");
                        this.hZonaComer.add(hormiga.getName()); // Queda reflejado que está dentro
                        this.cambioRelease();
                        Thread.sleep((long) (((Math.random())+1)*1000)); // Tiempo que tarda en depositarlo
                        this.unidadesComidaZonaComer.addAndGet(unidadesADepositar); // Deposita la
comida en el almacén
                        this.cambioRelease();
                        this.addLog(hormiga, '', ha depositado la comida y ha salido de la zona comer.");
                        this.semUnidadesZonaComer.release();
                } catch(InterruptedException ie) {}
                finally {
                        this.hZonaComer.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }
        // Métodos hormigas soldado
        public void entrarZonaInstruccion(HormigaSoldado hormiga) {
                try {
                        this.addLog(hormiga, '', ha entrado a la zona instruccion a entrenar.");
                        this.hZonaInstruccion.add(hormiga.getName()); // Queda reflejado que está dentro
                        this.cambioRelease();
                        Thread.sleep((long) (((Math.random()*6)+2)*1000)); // Tiempo que tarda en entrenar
                        this.addLog(hormiga, '', ha terminado de entrenar y ha salido de la zona de
instrucción.");
                        this.hZonaInstruccion.remove(hormiga.getNombreHormiga());
                } catch(InterruptedException ie) {
                        this.hZonaInstruccion.remove(hormiga.getNombreHormiga());
```

```
                        this.cambioRelease();
                        this.lucharInsectoInvasor(hormiga);
                }
                finally {
                        this.cambioRelease();
                }
        }
        public void lucharInsectoInvasor(HormigaSoldado hormiga) {
                this.pasarTunelSalida(hormiga);
                this.hRepeliendoInsecto.add(hormiga.getNombreHormiga());
                this.cambioRelease();
                try {
                        this.addLog(hormiga, ", se está enfrentando al insecto invasor."); // FOR THE
COLONY
                        Thread.sleep(20*1000); // Tiempo que está luchando contra el insecto.
                        this.addLog(hormiga, ", ha terminado de enfrentarse al insecto invasor.");
                } catch (InterruptedException e) {}
                finally {
                        this.hRepeliendoInsecto.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                        this.pasarTunelEntrada(hormiga);
                }


        }
        // Métodos hormigas cria
        public void entrarZonaRefugio(HormigaCria hormiga) {
                this.hRefugio.add(hormiga.getNombreHormiga());
                this.cambioRelease();
                try {
                        this.addLog(hormiga, ", ha entrado al refugio (sólo hormigas cria)."); // Entra en el
refugio
                        this.semLuchandoInsecto.acquire();
                        this.addLog(hormiga, ", ha salido del refugio (sólo hormigas cria).");
                } catch (InterruptedException e) {}
                finally {
                        this.hRefugio.remove(hormiga.getNombreHormiga());
```

```
                        this.cambioRelease();
                }
        }

        // Métodos generales
        public void entrarZonaComerConsumir(Hormiga hormiga, int tiempoComer) {
                try {
                        this.semUnidadesZonaComer.acquire(); // Sincronización, no deja entrar a no ser que haya unidades
                        this.addLog(hormiga, '', ha entrado a la zona comer a comer.");
                        this.hZonaComer.add(hormiga.getNombreHormiga()); // Queda reflejado que está dentro
                        this.unidadesComidaZonaComer.decrementAndGet();
                        this.cambioRelease();
                        Thread.sleep(tiempoComer*1000); // Tiempo que tarda en comer
                        this.addLog(hormiga, '', ha comido y ha salido de la zona comer.");
                } catch(InterruptedException ie) {
                        this.hZonaComer.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                        if(hormiga.getIdType() == "HS") { // Es una hormiga soldado a luchar
                                this.lucharInsectoInvasor((HormigaSoldado) hormiga);
                        }
                        if(hormiga.getIdType() == "HC") { // Es una hormiga cria al refugio
                                this.entrarZonaRefugio((HormigaCria) hormiga);
                        }
                }
                finally {
                        this.hZonaComer.remove(hormiga.getNombreHormiga());
                        this.cambioRelease();
                }
        }

        public void entrarZonaComerConsumir(Hormiga hormiga) {
                this.entrarZonaComerConsumir(hormiga, 3);
        }
```

```
	public void entrarZonaDescanso(Hormiga hormiga, int tiempoDescansando) {
		try {
			this.addLog(hormiga, '', ha entrado a la zona de descanso a mimir.");
			this.hZonaDescanso.add(hormiga.getNombreHormiga()); // Queda reflejado que
está dentro
			this.cambioRelease();
			Thread.sleep(tiempoDescansando * 1000); // Tiempo que tarda en comer
			this.addLog(hormiga, '', ha descansado y ha salido de la zona de descanso.");
		} catch(InterruptedException ie) {
			this.hZonaDescanso.remove(hormiga.getNombreHormiga());
			this.cambioRelease();
			if(hormiga.getIdType() == "HS") { // Es una hormiga soldado a luchar
				this.lucharInsectoInvasor((HormigaSoldado) hormiga);
			}
			if(hormiga.getIdType() == "HC") { // Es una hormiga cria al refugio
				this.entrarZonaRefugio((HormigaCria) hormiga);
			}
		}
		finally {
			this.hZonaDescanso.remove(hormiga.getNombreHormiga());
			this.cambioRelease();
		}
	}

	// Override de la interfaz Runnable
	@Override
	public void run() {
		// Me creo el hilo del log
		new Thread(this.log).start();

		try {
			new HormigaObrera(this).start();
			Thread.sleep((long) (((Math.random()*2.7)+0.8)*1000));
		} catch (LoggingException | InterruptedException e) {
```

```
                    e.printStackTrace();
                }
                try {
                    for(int i =0; i < 9999; ) {

                        if(this.pararEjecucion.get()) { // Espera activa :(
                            Thread.sleep(1000);
                        }
                        else {
                            // Cada 3 hormigas obreras se crerarán una soldado y una cria
                            if(i%4 == 3) {
                                new HormigaSoldado(this).start();
                            }
                            if(i%4 == 2) {
                                new HormigaCria(this).start();
                            }
                            else {
                                new HormigaObrera(this).start();
                            }
                            Thread.sleep((long) (((Math.random()*2.7)+0.8)*1000));
                            i++;
                        }
                    }
                } catch (LoggingException | InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

}
```

parte1.HormigaCria.java

```
package parte1;


import java.util.concurrent.atomic.AtomicInteger;
```

```
import parte1.Logging.LoggingException;
import parte1.interfaces.Hormiga;

public class HormigaCria extends Thread implements Hormiga{

	// Define sus propios valores de clase
	protected static String idType = "HC";
	protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHC");
	protected static AtomicInteger contador = new AtomicInteger(0);

	static {
		HormigaController.setHormigasCria(grupoHilos);
	}

	// Atributos especificos de cada hormiga obrera
	private int idHormiga;
	private HormigaController controlador;

	// Métodos implementados en la interfaz
	@Override
	public String getIdType() {
		return idType;
	}
	@Override
	public int getIdHormiga() {
		return this.idHormiga;
	}

	@Override
	public HormigaController getControlador() {
		return this.controlador;
	}

	@Override
```

```
public String getNombreHormiga() {
		return idType.concat(String.format("%04d", this.idHormiga));
}

// Métodos específicos para cada hormiga soldado
public HormigaCria(HormigaController hormigaController) throws LoggingException{
		super(grupoHilos, "");
		this.idHormiga = contador.getAndIncrement();
		this.controlador = hormigaController;
		super.setName(this.getNombreHormiga());
		try {
			this.controlador.addLog(this, ", creación de la hormiga cria.");
		} catch (InterruptedException e) {
			throw new Logging.LoggingException(e.getMessage());
		}
}

// Override de los métodos de la clase Thread
@Override
public void run() {
		// Comportamiento que harán todas las hormigas al nacer.
		try {
			this.controlador.addLog(this, ", acaba de nacer...");
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		this.controlador.pasarTunelEntrada(this);

		/*
		 * Dado que las hormigas cría son muy débiles, en caso de que exista la
		 * amenaza de un insecto invasor, todas ellas (tanto las creadas de forma
		 *  previa a la amenaza como las generadas durante la amenaza) deben
		 *  dejar lo que estén haciendo en ese momento y acudir rápidamente a la
		 *  zona de REFUGIO, hasta que la amenaza haya desaparecido, volviendo
		 *  entonces a su comportamiento habitual (iniciando su alimentación en la
```

```
                * ZONA PARA COMER)
                */
               if(HormigaController.getLuchandoInsectoInvasor().get()) {
                       this.controlador.entrarZonaRefugio(this);
               }

               while(true) {
                       if(this.controlador.getPararEjecucion().get()) { // Si está true, se suspende la hormiga.
                               this.suspend();
                       }
                       /*
                        * Accede a la ZONA PARA COMER, tardando un tiempo aleatorio de entre 3 y 5
                        * segundos en alimentarse. A continuación, descansan 4 segundos en la
                        * ZONA DE DESCANSO
                        */
                       this.controlador.entrarZonaComerConsumir(this, (int)((Math.random()*2) + 3));
                       this.controlador.entrarZonaDescanso(this, 4);
               }
       }
}
```

parte1.HormigaObrera.java

```
package parte1;

import java.util.concurrent.atomic.AtomicInteger;

import parte1.Logging.LoggingException;
import parte1.interfaces.Hormiga;

public class HormigaObrera extends Thread implements Hormiga{

       // Define sus propios valores de clase
       protected static String idType = "HO";
       protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHO");
       protected static AtomicInteger contador = new AtomicInteger(0);
```

```
static {
	HormigaController.setHormigasObreras(grupoHilos);
}

// Atributos especificos de cada hormiga obrera
private int idHormiga;
private HormigaController controlador;

// Métodos implementados en la interfaz
@Override
public String getIdType() {
	return idType;
}
@Override
public int getIdHormiga() {
	return this.idHormiga;
}

@Override
public HormigaController getControlador() {
	return this.controlador;
}

@Override
public String getNombreHormiga() {
	return idType.concat(String.format("%04d", this.idHormiga));
}

// Métodos de clase específicos de la hormiga obrera

// Métodos específicos para cada hormiga obrera
```

```
	public HormigaObrera(HormigaController hormigaController) throws LoggingException{
		super(grupoHilos, "");
		this.idHormiga = contador.getAndIncrement();
		this.controlador = hormigaController;
		super.setName(this.getNombreHormiga());
		try {
			this.controlador.addLog(this, ", creación de la hormiga obrera.");
		} catch (InterruptedException e) {
			throw new Logging.LoggingException(e.getMessage());
		}
	}

	// Override de los métodos de la clase Thread
	@Override
	public void run() {
		// Comportamiento que harán todas las hormigas al nacer.
		try {
			this.controlador.addLog(this, ", acaba de nacer...");
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		this.controlador.pasarTunelEntrada(this);
		// Conportamiento específico de cada hormiga.
		int i = 0;
		while(true) {
			if(this.controlador.getPararEjecucion().get()) { // Si está true, se suspende la hormiga.
				this.suspend();
			}
			/*Accede al ALMACÉN DE COMIDA para coger cinco elementos de comida requiriendo entre 1 y 2
			 * segundos. Entonces, viaja hacia la ZONA PARA COMER, tardando un tiempo aleatorio de
			 * entre 1 y 3 segundos. A continuación, accede a la ZONA PARA COMER y deposita el elemento
			 * de comida, consumiendo entre 1 y 2 segundos. Desde este momento, elemento de comida
			 * está disponible para ser ingerido por cualquier hormiga de la colonia.
```

```
                        */
                        if(this.getIdHormiga() % 2 == 0) { // Hormiga obrera par
                                this.controlador.entrarAlmacenComidaConsumir(this);
                                this.controlador.transicionZonaComer(this);
                                this.controlador.entrarZonaComerDepositar(this, 5);
                        }
                        /* Acude a la zona exterior de la colonia a coger cinco elementos de comida y
                         * lo lleva al interior de la colonia, tardando 4 segundos. A continuación,
                         * lo deposita en el ALMACÉN DE COMIDA, tardando un tiempo aleatorio de entre 2
                         * y 4 segundos. A este ALMACÉN DE COMIDA sólo pueden acceder
simultáneamente
                         * 10 hormigas
                         */
                        else { // Hormiga obrera impar
                                this.controlador.pasarTunelSalida(this); // Está fuera
                                this.controlador.transicionBuscandoComida(this); // Buscar comida fuera
                                this.controlador.pasarTunelEntrada(this); // Vuelve a entrar
                                this.controlador.entrarAlmacenComidaDepositar(this); // Guarda la comida
en el almacen
                        }
                        i++;
                        /*
                         * Todas las hormigas obreras, pares o impares, tras realizar 10 iteraciones
completas
                         * de su acción principal, pasan por la ZONA PARA COMER para reponer fuerzas y
comer,
                         * consumiendo 1 unidad de alimento y tardando 3 segundos. A continuación,
acceden a
                         * la ZONA DE DESCANSO, consumiendo allí 1 segundo, para, a continuación, volver
a
                         * retomar su actividad habitual.
                         */
                        if(i == 10) {
                                i = 0;
                                this.controlador.entrarZonaComerConsumir(this);
                                this.controlador.entrarZonaDescanso(this, 1);
                        }
```

```
                }
        }
}
```

parte1.HormigaSoldado.java

```
package parte1;


import java.util.concurrent.atomic.AtomicInteger;


import parte1.Logging.LoggingException;
import parte1.interfaces.Hormiga;


public class HormigaSoldado extends Thread implements Hormiga{


        // Define sus propios valores de clase
        protected static String idType = "HS";
        protected static ThreadGroup grupoHilos = new ThreadGroup("HilosHS");
        protected static AtomicInteger contador = new AtomicInteger(0);


        static {
                HormigaController.setHormigasSoldado(grupoHilos);
        }


        // Atributos especificos de cada hormiga obrera
        private int idHormiga;
        private HormigaController controlador;


        // Métodos implementados en la interfaz
        @Override
        public String getIdType() {
                return idType;
        }
        @Override
        public int getIdHormiga() {
                return this.idHormiga;
```

```
	}


	@Override
	public HormigaController getControlador() {
		return this.controlador;
	}


	@Override
	public String getNombreHormiga() {
		return idType.concat(String.format("%04d", this.idHormiga));
	}


	// Métodos específicos para cada hormiga soldado
	public HormigaSoldado(HormigaController hormigaController) throws LoggingException{
		super(grupoHilos, "");
		this.idHormiga = contador.getAndIncrement();
		this.controlador = hormigaController;
		super.setName(this.getNombreHormiga());
		try {
			this.controlador.addLog(this, ", creación de la hormiga soldado.");
		} catch (InterruptedException e) {
			throw new Logging.LoggingException(e.getMessage());
		}
	}


	// Override de los métodos de la clase Thread
	@Override
	public void run() {
		// Comportamiento que harán todas las hormigas al nacer.
		try {
			this.controlador.addLog(this, ", acaba de nacer...");
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
```

```
                this.controlador.pasarTunelEntrada(this);
                int i = 0;
                while(true) {
                        if(this.controlador.getParar Ejecucion().get()) { // Si está true, se suspende la hormiga.
                                this.suspend();
                        }
                        /*
                         * Hace instrucción en la ZONA DE INSTRUCCIÓN, tardando un tiempo aleatorio de
                         * entre 2 y 8 segundos en hacer esta operación. A continuación, descansa 2
                         * segundos en la ZONA DE DESCANSO.
                         */
                        this.controlador.entrarZonaInstruccion(this);
                        this.controlador.entrarZonaDescanso(this, 2);
                        i++;
                        /*
                         * Todas las hormigas soldado, tras realizar 6 iteraciones completas de su acción principal,
                         * pasan por la ZONA PARA COMER a reponer fuerzas y comer, consumiendo 1 unidad de
                         * alimento. Esta operación tarda 3 segundos en realizarse.
                         */
                        if(i == 6) {
                                i = 0;
                                this.controlador.entrarZonaComerConsumir(this);
                        }
                }
        }
}
```

parte1.Logging.java

```
package parte1;

import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.BlockingQueue;
```

```
public class Logging implements Runnable{

	public static class LoggingException extends Exception{
		private static final long serialVersionUID = -3604260959065292002L;

		LoggingException(){
			super("No se ha configurado la cola del logging y es obligatoria para la ejecución de los hilos.");
		}

		LoggingException(String mensaje){
			super(mensaje);
		}
	}

	private BlockingQueue<String> listaLog;

	public BlockingQueue<String> getListaLog() {
		return listaLog;
	}

	public void setListaLog(BlockingQueue<String> listaLog) {
		this.listaLog = listaLog;
	}

	private String nombreFichero = "evolucionColonia.txt";

	public String getNombreFichero() {
		return nombreFichero;
	}

	public void setNombreFichero(String nombreFichero) {
		this.nombreFichero = nombreFichero;
```

```
	}


	public Logging(BlockingQueue<String> lista) {
		this.listaLog=lista;
	}


	public Logging(BlockingQueue<String> lista, String nombreFichero) {
		this.listaLog=lista;
		this.nombreFichero = nombreFichero;
	}


	@Override
	public void run() {
		try (FileWriter out = new FileWriter(nombreFichero)) {
			while(true) {
				out.write(listaLog.take()+"\n");
				out.flush();
			}
		} catch (InterruptedException | IOException e) {e.printStackTrace();}
	}


	// Método de clase
	public static String getMensaje(String s) {
		s+="  ";
		Date fechaActual = new Date();
		String formato = "dd/MM/yyyy-HH:mm:ss:SSS";
		SimpleDateFormat sdf = new SimpleDateFormat(formato);
		return s.concat(sdf.format(fechaActual));
	}
}
```

PECLServidor.java

```
package emilio.peclservidor;


import java.awt.EventQueue;
```

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import java.awt.EventQueue;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import parte1.GUI;
import parte1.HormigaController;

public class PECLServidor {


	public static void main(String[] args) {
		try {
			// Inicializo el controlador
			HormigaController controlador = new HormigaController();
			new Thread(new ThreadGroup("HilosControladores"), controlador).start();
			//Arranca el espacio de nombres RMIRegistry local en el puerto 1099. No es
obligatorio
			Registry registry = LocateRegistry.createRegistry(1099);
			//Hace visible el objeto para clientes
			Naming.rebind("//127.0.0.1/ControladorHormiga", controlador);
			System.out.println("El controlador ha quedado registrado.");


			/**
			 * Launch the GUI application.
			 */

			EventQueue.invokeLater(new Runnable() {
				public void run() {
					try {
						GUI window = new GUI(controlador);
						window.getFrmAplicacinHormigas().setVisible(true);
					} catch (Exception e) {
```

```
                                                    e.printStackTrace();
                                                }
                                            }
                                        });


                                        System.out.println("Se ha creado la GUI.");
                                } catch (Exception e) {e.printStackTrace();}


                        }

}
```

parte1.interfaces.Hormiga.java

```
package parte1.interfaces;


import parte1.HormigaController;


public interface Hormiga {


        // Atributos comunes a todas las hormigas.
        public String getIdType();
        public int getIdHormiga();
        public HormigaController getControlador();
        public String getNombreHormiga();
}
```

parte1.interfaces.VistaHormigaGUI.java

```
package parte1.interfaces;


import java.util.Set;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;


/**
 * @author Emilio Macías Do Santos
 *
 *          Esta interfaz sirve para que la vista GUI.java pueda saber qué métodos llamar al controlador,
```

```
 *          de esta forma se implementa el MVC.
 *
 *          La vista no verá nada más que los métodos que deba saber.
 */
public interface VistaHormigaGUI{
        // Getter y Setters públicos
        public Set<String> gethDentroColonia();

        public Set<String> gethFueraColonia();

        public Set<String> gethBuscandoComida();

        public Set<String> gethRepeliendoInsecto();

        public Set<String> gethAlmacenComida();

        public Set<String> gethLlevandoZonaComer();

        public Set<String> gethZonaInstruccion();

        public Set<String> gethZonaDescanso();

        public Set<String> gethZonaComer();

        public Set<String> gethRefugio();

        public int getUnidadesComidaAlmacen();

        public AtomicInteger getUnidadesComidaZonaComer();

        public AtomicBoolean getPararEjecucion();

        // Métodos para parar la ejecucion de las hormigas dando al botón Parar
        public void pararEjecucion();
```

```
	public void continuarEjecucion();
	// Método para generar un insecto invasor
	public void generarInsectoInvasor();

	// Métodos que utilizan un semáforo para controlar los eventos.
	public void cambioAdquire() throws InterruptedException;

	public void cambioRelease();

}
```

parte2.interfaces.VistaClientesRMI.java

```
package parte2.interfaces;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Set;

/**
 * @author Emilio Macías Do Santos
 *
 *	Esta interfaz sirve para que la vista GUI.java pueda saber qué métodos llamar al controlador,
 *	de esta forma se implementa el MVC.
 *
 *	La vista no verá nada más que los métodos que deba saber.
 */
public interface VistaClienteRMI extends Remote{
	// Getter y Setters públicos
	public Set<String> gethDentroColoniaRMI() throws RemoteException;

	public Set<String> gethFueraColoniaRMI() throws RemoteException;

	public Set<String> gethZonaInstruccionRMI() throws RemoteException;

	public Set<String> gethRepeliendoInsectoRMI() throws RemoteException;
```

```
        public Set<String> gethZonaComerRMI() throws RemoteException;


        public Set<String> gethRefugioRMI() throws RemoteException;


        // Método para generar un insecto invasor
        public void generarInsectoInvasorRMI() throws RemoteException;


        // Métodos que utilizan un semáforo para controlar los eventos.
        public void cambioAdquireRMI() throws InterruptedException, RemoteException;


        public void cambioReleaseRMI() throws RemoteException;


}
```