Grado en Ingeniería Informática

Curso 2022/2023 – Convocatoria Ordinaria

03233102S – Gutiérrez Sánchez, Carlos

00000000X – Nieto Arias, Pablo

ÍNDICE

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

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

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

4) diagrama de clases; y

5) el código fuente, como anexo

Análisis del problema

En el enunciado se describe el funcionamiento de una colonia de hormigas. Hay tres tipos de hormigas: las obreras, las soldados y las crías. Estos serán los primeros actores, son elementos activos del código. Se menciona en el enunciado

Diseño general del sistema

**Clases principales**

**Clases main:**

Servidor.java

Cliente.java

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

**Clases de hilos:**

Cria.java

Hormiga.java

Obrera.java

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

Soldado.java

**Clases con variables compartidas:**

Almacen.java

Bicho.java

Comedor.java

Descanso.java

Hormiguero.java

Instruc.java

Refugio.java

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

Escritor.java

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

TareaEscribir.java

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

Clases responsables de mostrar las ventanas (JFrames)

VentanaPrincipal.java

VentanaRemota.java

**Clases responsables de gestionar la conexión:**

InterfazOperaciones.java

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

Operador.java

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

Servidor.java

Es la misma que la ya comentada al principio de esta sección