Grado en Ingeniería Informática

Curso 2021-2022

09064995M – Marco González Martínez

03217363P – Sergio Lorenzo Montiel

Índice

1. Análisis de alto nivel

- El proyecto tendrá que incorporar una aplicación de servidor que realizará una simulación del campamento, y una aplicación de cliente que consultará algunos datos del campamento.
- El campamento debe tener dos posibles entradas de acceso.
- Los monitores que lleguen elegirán (aleatoriamente) una de las dos puertas, la abrirán en caso de estar cerrada, se asegurarán de que ambas entradas del campamento queden abiertas (forzando la elección si es el último monitor y todavía queda una puerta cerrada).
- Los niños elegirán una puerta de acceso, si el campamento se encontrase lleno, o las puertas estuviesen cerradas, esperarán en una cola para poder entrar.
- Una vez el campamento esté lleno, si hay niños esperando en ambas puertas, entrarán en cuanto otro niño abandone el campamento, respetando una alternancia.
- Los niños del programa estarán implementados como hilos. Se creará un hilo nuevo de este tipo con un intervalo entre 1 y 3 s (aleatoriamente) hasta que se hayan creado 20 000 niños.
- Existen 3 actividades del campamento (Tirolina, Merendero y Soga), que los niños podrán seleccionar (aleatoriamente).
- Existe una Zona Común en la que los hilos esperarán un tiempo determinado.
- Los Niños serán identificados con un texto del siguiente formato: “NXXXXX” siendo la X el número que representa únicamente el orden en el que se han generado los niños.
- Los niños realizarán 15 actividades en el campamento, una vez acabadas, abandonarán el campamento.
- Los niños descansarán en la Zona Común entre cada actividad un tiempo aleatorio entre 2 y 4 s.
- Los niños no podrán elegir merendero sin antes haber realizado 3 actividades de los otros tipos como mínimo.
- En la actividad tirolina, los niños esperarán primero en una cola. Podrán pasar cuando el monitor esté en la actividad, y no haya ningún otro niño utilizando la tirolina.
- La actividad tirolina tiene una fase de preparación en la que se invierte 1s; una fase de tirarse de 3s; y de finalización de 0,5s tras la cual abandonan la actividad.
- En la actividad Soga, los niños esperarán en la cola hasta que llegue un monitor, el monitor irá asignándolos a los distintos equipos.
- La actividad comenzará cuando haya 5 niños en cada uno de los dos equipos. Durará 7 segundos, y acabará con un equipo ganador cuyos niños acumularán 2 actividades, y un equipo perdedor cuyos niños acumularán 1 actividad.
- Si un niño elige la actividad de Soga, pero no queda espacio libre, desistirá y elegirá otra actividad del campamento.
- La actividad Merienda tendrá un aforo de 20 niños, que accederán con criterio FIFO.
- En la zona del merendero hay una pila de bandejas listas y otra de bandejas sucias, de dónde los niños cogerán bandejas listas para comer (tardando 7s en comer), y devolverán la bandeja usada a la pila de bandejas sucias.
- Los monitores del proyecto estarán implementados como hilos
- Los Monitores serán identificados con un texto del siguiente formato: “MX” siendo la X el número que representa el orden en el que se han generado los monitores.
- Existirán 4 monitores, cada uno elegirá una actividad, asegurándose de que todas tienen monitores suficientes tras la elección, y dirigirá siempre la misma actividad.

- En la actividad Merienda deben acudir 2 monitores. En las otras 2 actividades sólo 1 monitor.
- Los monitores descansarán en la Zona Común cada 10 actividades un tiempo entre 1 y 2 segundos. Tras ello, volverán a su puesto de trabajo.
- Una vez el monitor llegue a la actividad Tirolina los niños podrán empezar a utilizarla.
- En la actividad Soga, el Monitor se encargará de ir asignando los niños a los equipos, una vez haya 5 niños en cada equipo el juego comenzará e indicará el equipo ganador.
- En la actividad Merienda los monitores se encargan de limpiar y preparar las bandejas sucias (entre 3 y 5s) y dejarlas en la pila de listas.
- El comportamiento del campamento se registrará en un fichero log llamado "evolucionCampamento.txt" de forma que los elementos del programa puedan ir añadiendo la información al documento de forma segura.
- Existirá una interfaz gráfica en la que se mostrará a tiempo real el funcionamiento del campamento, mostrando información sobre donde se encuentra cada niño y monitor, entre otras.
- Esta interfaz deberá permitir al usuario pausar o reanudar el sistema.
- La aplicación cliente tendrá una interfaz gráfica que permite al usuario realizar consultas sobre ciertos datos del sistema.
- Se podrán consultar los siguientes datos:
    - Número de niños esperando en la cola de Tirolina.
    - Número de niños esperando en la cola de Soga.
    - Número de niños en la zona Merienda
    - Número de usos de la actividad Tirolina
    - Número de bandejas listas y sucias.
    - Número de actividades de un niño en concreto, a partir de introducir su identificador.
- Adicionalmente, se han incorporado funcionalidades que amplían las instrucciones indicadas:
- Al abandonar el campamento, los niños califican el campamento con una nota aleatoria.
- El campamento almacenará un valor medio de las notas recibidas según sea calificado nuevamente.
- Se podrá consultar la valoración media del campamento desde el cliente del sistema.

2. Diseño general del sistema y de herramientas de sincronización

- Hemos separado en paquetes la funcionalidad de servidor y cliente de manera que la del cliente estará en el paquete Cliente y la del servido estará en los paquetes Interfaz y Modelo.
- La conexión entre cliente y servidor se ha realizado mediante el uso de sockets TCP.
- En generador se irán generando los hilos monitor, los hilos para escuchar las consultas y los hilos de niños. Se generarán 4 monitores y 20000 niños, estos últimos se irán generando cada intervalo de entre 1 y 3 segundos.
- Para tratar la entrada del campamento se han diseñado dos colas de entrada (una para cada puerta) declaradas como CopyOnWriteArrayList para lograr exclusión mutua a la hora de modificarlas.

- Para la apertura de las puertas se han definido dos métodos sincronizados, uno para cada puerta, para garantizar la exclusión mutua de forma que varios monitores no abran la misma puerta. Si tres monitores han elegido una de las dos puertas para entrar al campamento se forzará al último de ellos entrar por la otra puerta para así abrirla.
- Con la ayuda de un mecanismo CountDownLatch, los niños se quedarán bloqueados en la cola de entrada hasta que llegue un monitor a abrirla y acto seguido entrarán al campamento.
- Para el caso de que el campamento esté lleno se trata el aforo y la alternancia mediante un Lock y Condition. Cuando un niño quiere entrar al campamento y aforo está completo esperará por el Condition asociado a la puerta en la que está esperando mientras no haya huecos disponibles. Cuando un niño salga del campamento, dependiendo de la alternancia actual, ejecutará un signal a la puerta que corresponda dejando paso al niño que esté esperando por entrar.
- Una vez el niño ha entrado, este se añadirá al registro de niños que han estado en el campamento (se utilizará más adelante para las consultas del Cliente), escogerá actividades del campamento hasta que realice 15 actividades y después de esto abandonará el campamento.
- Una vez el monitor ha entrado al campamento se le asignará una actividad que no tenga todavía suficientes monitores. Esto se ha controlado con un método synchronized que reserva la actividad y un ArrayList de enteros que representan a cada actividad y su utilidad será comprobar si la actividad está disponible para elegir. Cuando una actividad tiene monitores suficientes desaparecerá del ArrayList de manera que otros monitores no puedan escogerla.
- Los espacios de descanso de la Zona Común tanto de los niños como de los monitores están declarados como dos CopyOnWriteArrayList y mediante los métodos descansar se añadirán a su zona de descanso, ejecutarán su respectivo sleep y después abandonarán esa zona de descanso.
- Los niños cada vez que terminen una actividad pasarán a la Zona Común donde se paralizarán (ejecutarán un sleep) entre 2 y 4 segundos y tras esto pasarán a realizar otra actividad.
- Indefinidamente los monitores estarán controlando las actividades y cada diez de estas accederán a la Zona Común donde el hilo monitor se paralizará (realizará un sleep) entre 1 y 2 segundos, volviendo tras esto a la actividad a la que estaba asignado.
- Para la cola de los niños de Tirolina se ha implementado una ConcurrentLinkedQueue para poder editarla de manera correcta. Se ha utilizado un Lock para controlar que solo un niño entre a la zona de Tirolina y dos Condition, uno para comprobar si hay monitor y otro para comprobar si es el primero de la cola, y en caso de que no cumplan una de las condiciones esperarán a ser notificados.
- Una vez el niño pasa a la zona de tirolina se le elimina de la cola y mediante un entero se controlará en que parte de la tirolina se encuentra el niño: si es 0 no habrá niño; si es 1, el niño estará en la zona de preparación ejecutando un sleep de 1 segundo; si es 2, el niño estará usando la tirolina ejecutando un sleep de 3 segundos y si el estado es 3 el niño se estará bajando de la tirolina ejecutando un sleep de 0,5 segundos. Tras salir de la Tirolina notificará al siguiente niño para que pueda utilizarla.
- El monitor de Tirolina una vez llegue notificará a los niños de que ha llegado permitiendo al primer niño poder utilizarla. Tras esto esperará a que 10 niños utilicen

la Tirolina mediante un Condition y acto seguido la abandonará para poder ir a Zona Común a descansar y luego volver a Tirolina.

- Al llegar a Soga, el niño comprobará cuantos niños hay en la actividad y si no hubiese espacio para él no entrará a la actividad Soga.
- En Soga la cola de espera para los niños se ha implementado mediante un ArrayBlockingQueue en la cual esperarán a que el monitor les asigne un equipo. Además, mediante un CyclicBarrier, esperarán a que el monitor finalice el juego.
- En cuanto al monitor en Soga, en primer lugar, irá tomando a los niños de la cola y los pondrá en uno de los dos equipos aleatoriamente a no ser que el equipo elegido estuviese lleno, en cuyo lugar seleccionará el equipo opuesto. Una vez estén los dos equipos llenos jugarán durante 7 segundos (mediante un sleep) y acto seguido decide qué equipo es el ganador quitándole 2 actividades a realizar a los niños del equipo ganador y 1 a los del equipo perdedor, eliminándoles en el proceso de la actividad.
- Finalmente, el monitor llega al CyclicBarrier liberando a todos los hilos de forma que los niños irán a elegir otra actividad.
- El proceso ejecutado por el monitor se llevará a cabo hasta que haya realizado 10 partidas, tras las que abandonará la actividad e irá a descansar a la Zona Común.
- Para limitar el aforo del merendero se ha implementado un semáforo fair (FIFO) de 20 permisos.
- Una vez el niño adquiere un permiso para entrar accede a la zona de merienda, intentará coger una de las bandejas listas, en caso de que no haya esperará mediante un wait y esperará a ser notificado cuando se añada una nueva bandeja lista. Una vez la haya cogido merendará durante 7 segundos (mediante un sleep), después dejará la bandeja en la lista de sucias y finalmente dejará un hueco libre en el merendero realizando un release en el semáforo.
- Los monitores en Merendero extraerán bandejas sucias si hay para poder limpiarlas y si no esperarán a ser notificados cuando se añada una bandeja sucia. Una vez extraída tardarán en limpiarla entre 3 y 5 segundos (mediante un sleep) y una vez se haya limpiado se añadirá a la lista de limpias. Al haber limpiado 10 bandejas abandonarán el Merendero e irán a la Zona Común a descansar para luego volver a su puesto.
- Con el fin de almacenar el comportamiento progresivo del programa, se ha diseñado un hilo Escritor, accesible por todas las clases del campamento. Este hilo consta de un búfer de mensajes (String) en el cual otros hilos añadirán mensajes, y el hilo escritor se encargará de serializarlos a un archivo de texto, siguiendo el modelo de sincronización productor-consumidor.
- Para poder mantener actualizada la interfaz de la aplicación, se ha creado un hilo Pintor, que tiene acceso a los datos almacenados en el campamento, y a los elementos de la ventana mostrada al usuario. De esta forma, en un bucle indefinido, con una espera de 50 ms, recoge los datos correspondientes del campamento y los representa en la interfaz.
- Para permitir la opción de pausar y reanudar la ejecución del programa, se utiliza una clase Paso accesible por todas las clases del campamento. Esta clase Paso consta de un monitor en el que los hilos se quedan bloqueados cuando se ha pulsado la opción de detener el programa; al reanudarlo, se notifica a todos los hilos bloqueados para que reanuden su ejecución.
- La interfaz de cliente permitirá hacer diferentes consultas en función del botón que se pulse. Cada botón de consultar mostrará la información correspondiente. En el caso de

mostrar las actividades realizadas por un niño se deberá escribir en el cuadro de texto el niño en concreto.

- Una vez se ha pulsado un botón se crea un hilo consulta para evitar que el sistema se quede bloqueado a la espera de datos de red. Este hilo se encargará de enviar un entero que represente la información solicitada, y si es la consulta de las actividades de un niño, también enviará el niño que se busca.
- La clase Respuesta recibe la información de la consulta que se busca realizar y obtiene la información correspondiente de Campamento para enviársela al Cliente. El servidor se ha diseñado de forma multicliente mediante esta clase haciendo que herede de Thread, y lanzando un hilo nuevo para cada Consulta escuchada.
- El Registro de niños, se ha implementado como una clase adicional contiene ordenadamente a los niños del sistema por su id, para luego poder realizar búsqueda binaria. Esto es necesario para que la consulta de un niño por su id sea ágil.

3. Clases principales

a. Campamento

Esta clase representa el campamento, la clase contiene cada actividad disponible y algunos datos sobre el campamento.

Atributos:

nMonitoresMerienda -> Entero que representa cuantos monitores ya han elegido la actividad Merienda.
nMonitoresTirolina -> Entero que representa cuantos monitores ya han elegido la actividad Tirolina.
nMonitoresSoga -> Entero que representa cuantos monitores ya han elegido la actividad Soga.
nMonitoresDesMer -> Entero que indica la cantidad de monitores que deben acudir a la actividad Merienda.
nMonitoresDesTir -> Entero que indica la cantidad de monitores que deben acudir a la actividad Tirolina.
nMonitoresDesSoga -> Entero que indica la cantidad de monitores que deben acudir a la actividad Soga.
nCalificaciones -> Entero que representa cuantas calificaciones ha recibido el campamento.
media -> Real que indica la calificación media que ha recibido el campamento.
actividades -> ArrayList de enteros que representan las actividades del campamento, sirve para el momento de repartir las actividades entre los monitores.
entrada -> Instancia de la clase Entrada que representa la entrada al campamento.
tirolina -> Instancia de la clase Tirolina que representa la zona tirolina.
soga -> instancia de la clase Soga que representa la zona soga.
merendero -> Instancia de la clase merendero que representa la zona merendero.
zonaComun -> Instancia de la clase ZonaComun que representa la zona común del campamento.
registroNinno -> Instancia de la clase RegistroNinno que se utilizará para

almacenar datos de todos los niños que entran al campamento.

Métodos:

Varios métodos de esta clase son métodos que únicamente llaman en cascada a métodos de las clases contenidas.

calificar(Ninno ninno) -> Genera una calificación aleatoria para el campamento, modificando la nota media del campamento.

elegirEntrada(Monitor monitor) -> Le asigna al monitor que invoca el método una entrada para que acceda al campamento, esto se hará de forma aleatoria a no ser que el último monitor vaya a dejar una de las puertas sin abrirse.

reservaActividad(Monitor monitor) -> Permite el reparto de actividades entre los monitores, haciendo que todas las actividades acaben con monitores suficientes.

b. Entrada

Esta clase representa la entrada del campamento.

Atributos:

huecosDisponibles -> Entero que representa los huecos disponibles del campamento

nMonitoresP1 -> AtomicInteger que representa el total de monitores que han pasado por la puerta 1.

nMonitoresP2 -> Igual que el anterior pero de la puerta 2.

alternancia -> Booleano que representa la alternancia que se va a seguir con las puertas cuando el campamento se llene. Si está a false pasará el siguiente niño que se encuentre en la puerta 2 y si es true pasará el de la puerta 1.

colaEntrada1 -> CopyOnWriteArrayList que representa la cola de entrada de la puerta 1.

colaEntrada2 -> Igual al anterior pero de la puerta 2.

cdl1 -> CountDownLatch utilizado para que una vez pase el primer monitor por la puerta 1 y la abra los niños de esa puerta puedan empezar a pasar.

cdl2 -> Igual al anterior pero aplicado a la puerta 2.

lockEntrada -> Lock utilizado para garantizar exclusión mutua a la hora de que los niños entren y salgan del campamento.

puerta1 -> Condition asociado a lockEntrada utilizado para bloquear a los niños de la puerta 1 cuando no hay huecosDisponibles es igual a 0 (no hay huecos disponibles).

puerta2 -> igual al anterior pero asociado a la puerta 2.

Métodos:

entrarPuerta1(Ninno ninno) -> En primer lugar añade al niño a la cola de entrada de la puerta 1 y espera a que la puerta sea abierta. Una vez abierta, adquiere lockEntrada para evitar que otro niño entre antes que él y comprueba si hay huecos disponibles, en caso de no haberlos ejecutará el await del Condition puerta1. Si pudiera pasar se le retiraría de la cola de puerta 1, restaría 1 a huecosDisponibles y liberaría lockEntrada para que otro niño pueda entrar.

entrarPuerta2(Ninno ninno) -> Similar a entrarPuerta1, pero en relación con la puerta 2, utilizando el Condition puerta2 en caso de no haber huecos disponibles.

salirCampamento(Ninno ninno) -> Recibe como parámetro de entrada un niño.

Una vez adquiere lockEntrada añade un espacio disponible y comprueba si hay niños esperando en las dos colas. En caso de ser así comprobará cual es la alternancia y dependiendo de si el valor es true o false mandará un signal a puerta1 o puerta2 respectivamente para notificar que el siguiente niño (bloqueado por su respectivo await) pueda pasar al campamento.
abrirCamp1(Monitor monitor) -> Recibe como parámetro de entrada un monitor. Una vez el monitor comprueba que nadie más ha abierto la puerta (cdl1.getCount() es mayor que 0), espera entre 0,5 y 1 segundos mediante un sleep y realiza un countDown sobre cdl1 permitiendo así entrar a los niños.
abrirCamp2(Monitor monitor) -> Similar a abrirCamp1, pero sobre la puerta dos utilizando cdl2.
IncrementNMonitoresP1() -> incrementa en 1 nMonitoresP1.
IncrementNMonitoresP2() -> incrementa en 1 nMonitoresP2.

c. ContadorBandejas

Esta clase se utiliza para controlar las bandejas, se trata en exclusión mutua.

Atributos:

cantidad -> Entero que indica la cantidad de bandejas que hay.

Métodos:

annadirBandeja() -> Añade uno al contador de bandejas y notifica por si alguien está esperando a una bandeja.
extraerBandeja() -> Si hay bandejas para extraer quita uno al contador de bandejas, si no esperará a que haya una para extraer.

d. Merendero

Esta clase representa la zona de merendero del campamento.

Atributos:

bandLimpias, bandSucias -> Objetos de la clase ContadorBandejas, que permitirán tratar de manera concurrente, la cantidad de bandejas limpias y sucias respectivamente.
colaMerendero -> Cola concurrente que contiene los niños que esperan para poder entrar al merendero.
ninnoMerendero -> CopyOnWriteArrayList que contiene a los niños que hay actualmente en el merendero.
monMerendero -> CopyOnWriteArrayList que contiene a los monitores que hay actualmente en el merendero.
semMerienda -> Semáforo fair que se utilizará para controlar el aforo del merendero.

Métodos:

merendar(Ninno ninno) -> Realiza la acción de la actividad merienda para el hilo que invoca el método. Utiliza semMerienda para controlar el aforo de la actividad, extrae una bandeja de bandLimpias, y cuando termina la actividad la devuelve la bandeja añadiendo una a bandSucias.
merendero(Monitor mon) -> El hilo que invoca al método, realizará su tarea en la actividad merendero, extraerá bandejas de bandSucias, para añadirlas en bandLimpias, descansando cada 10 bandejas limpias.

e. Tirolina

Esta clase representa la zona de tirolina del campamento.

Atributos:

estadoTirolina -> Entero que representará el estado de la tirolina en ese momento (explicado en el punto 2).

vecesUsado -> Entero que representará las veces que se ha utilizado la tirolina.

ninnoTirolina -> Lista que contendrá el niño que esté utilizando la tirolina.

monTirolina -> Lista que contendrá el monitor que se encuentre en la zona de tirolina.

colaTirolina -> Cola concurrente que contiene a los niños que están esperando para usar la tirolina.

lockTirolina -> Lock utilizado para garantizar exclusión mutua a la hora de que los niños utilicen la tirolina.

monitorTirolina -> Condition asignado a lockTirolina utilizado para comprobar si hay un monitor en la tirolina.

primeroCola -> Condition asociado a lockTirolina utilizado para comprobar si un niño es el primero de la cola.

actividadesMonitor -> Condition asociado a lockTirolina utilizado cuando el monitor ha realizado 10 actividades.

Métodos:

tirolina(Ninno ninno) -> Realiza la acción de utilizar la tirolina. Al adquirir el lockTirolina y tras comprobar que hay monitor y es el primero de la cola, el niño va pasando por los diferentes estados de la tirolina hasta finalizar cuando se restará 1 a actividades restantes tanto a él como al monitor.

tirolina(Monitor monitor) -> Realiza la acción de notificar a los niños para usar la tirolina y esperará a que ellos realicen 10 actividades mediante el Condition actividadesTirolina. Una vez se realicen esas 10 actividades el monitor abandonará la zona de tirolina.

f. Soga

Esta clase representa la zona de soga del campamento.

Atributos:

tamEquipo -> entero que representa el tamaño que tendrán los equipos de la actividad.

colaSoga -> ArrayBlockingQueue de niños, en el que se contienen los niños que están esperando para realizar la actividad.

colaSogaEquipoA -> ArrayBlockingQueue de niños que contiene los niños que jugarán en el Equipo A.

colaSogaEquipoB -> ArrayBlockingQueue de niños que contiene los niños que jugarán en el Equipo B.

monSoga -> ArrayList de monitor que contiene el monitor de la actividad soga.

barreraSoga -> CyclicBarrier utilizada para lograr sincronización en la actividad.

Métodos:

soga(Ninno ninno) -> Añade al niño en la actividad si tiene hueco para jugar, sino simplemente sale del método. Cuando entra en la actividad, esperará en la barreraSoga para que el monitor, controle el juego.

soga(Monitor mon) -> El monitor, extrae niños de la cola de espera, para añadirlos en algún equipo, de forma que llenará las colas de los equipos. Se

aprovecha el uso de colas bloqueantes para que en caso de no haber niños suficientes en la cola espera hasta que haya uno, y poder añadir el niño sólo a equipos que todavía no tengan participantes suficientes. Después elige el equipo ganador, y sustrae las actividades correspondientes a cada niño (2 a los ganadores, 1 a los perdedores). Finalmente, esperará en la barreraSoga de forma que cuando haya llegado el monitor y todos los niños, los niños acaben la actividad.

g. ZonaComun

Esta clase representa la zona común del campamento.

Atributos:

ninnoZonaComun -> CopyOnWriteArrayList que contiene a los niños que se encuentran en la zona común.

monZonaComun -> CopyOnWriteArrayList que contiene a los monitores que se encuentran en la zona común.

Métodos:

descansar(Ninno ninno) -> Inserta al niño en la lista ninnoZonaComun, realiza un sleep de entre 2 y 4 segundos y lo elimina de la lista.

descansar(Monitor mon) -> Inserta al monitor en la lista monZonaComun, realiza un sleep de entre 1 y 2 segundos y lo elimina de la lista

h. Ninno

Esta clase que hereda de hilo representa a los niños que entrarán al campamento.

Atributos:

id -> Cadena que representa el identificador del niño.

contActividades -> Entero que representa cuantas actividades faltan por realizar.

totalActividades -> Entero que representa el total de actividades a realizar

campamento -> Instancia que representa la variable compartida de tipo Campamento.

Métodos:

entrarCamp() -> Este método, mediante un booleano aleatorio, llamará al método entrarPuerta1 o entrarPuerta2 de la clase Campamento para entrar al campamento.

seleccionarActividad() -> Realiza la selección de una actividad de forma aleatoria. Si en un momento dado faltan más de 12 actividades por realizar solo podrá escoger entre Soga o Tirolina. Una vez le falten 12 o menos actividades se añadirá Merendero a las opciones. Tras llamar a uno de los métodos de usar de la clase Campamento llamará a usarZonaComún de Campamento para descansar.

salirCamp() -> Realiza la salida del campamento llamando a los métodos de la clase Campamento salirCampamento y calificar.

substractActividad() -> Resta en uno contActividades.

run() -> Representa el ciclo de vida del niño. Primero llama al método entrarCamp, después hasta que no haya realizado 15 actividades llamará al método seleccionarActividad y tras esto llamará al método salirCamp.

i. Monitor

Esta clase que hereda de hilo representará a los monitores que controlan el campamento.

Atributos:

id -> Cadena que representa el identificador del monitor

campamento -> Instancia que representa la variable compartida de tipo Campamento.

nMonitores -> Entero que representa el total de monitores que va a haber.

contadorActividades -> Entero que representa cuantas actividades faltan antes del descanso.

actividadesHastaDescanso -> Entero que representa las actividades a realizar antes del descanso.

Métodos:

abrirEntrada() -> Mediante un booleano aleatorio llama al método de Campamento elegir entrada y dependiendo del valor que esa función haya devuelto llamará al método abrirCamp1 o abrirCamp2 de la clase Campamento.

substractActividad() -> resta en 1 contadorActividades.

run() -> Representa el ciclo de vida del monitor. Primero llama al método abrirEntrada() y acto seguido llama al método de Campamento reservarActividad. Dependiendo de que actividad se le asigne, de manera indefinida, llamará al respectivo método acceder de la actividad y posteriormente al método accederZonaComún de Campamento. Una vez haya ido a Zona Común se restaurarán las actividades a realizar antes del descanso.

j. RegistroNinno

Esta clase se encargará de almacenar todos los niños que entren al campamento

Atributos:

listaNinnos -> ArrayList de niños, contendrá todos los niños que pasen en algún momento.

cerr -> Lock que será utilizado para garantizar exclusión mutua en la modificación de la lista de niños.

Métodos:

annadir(Ninno ninno) -> Añade el niño recibido cómo parámetro al registro, asegurándose de dejarlo ordenado según su id.

binarySearch(String idNinno) -> realiza una búsqueda binaria a partir del id del niño.

numActividadesNinno(String idNinno) -> obtiene el número de actividades realizadas por el niño de id introducido como parámetro.

k. Paso

Esta clase se encargará de realizar las detenciones y reanudaciones en el funcionamiento del sistema.

Atributos:

detenido -> Booleano que representa si el sistema está detenido o no.

lock -> Lock empleado para lograr un Monitor de Hoare.

detener -> Condition asociado a lock empleado para realizar la detención del sistema.

Métodos:

mirar() -> Este método comprobará cual es el valor de detenido. Si está en false no hará nada más, pero si estuviera en true lo manda a la cola de condición del Monitor de Hoare.

reanudar() -> Este método pone en false a la variable detenido y realiza un signalAll para que los hilos salgan de la cola de condición.

detener() -> Este método pone en true a la variable detenido.

l. Generador

Esta clase que hereda de hilo se encargará de crear y lanzar los hilos de niño y monitor además de encargarse de lanzar los hilos de respuesta del servidor.

Atributos:

n_monitores, n_ninnos, n_actividades_ninnos,n_actividades_monitores -> enteros que representan número de niños, monitores, y las actividades que tendrán que realizar cada uno.

campamento -> Se trata de la instancia de la clase Campamento que será la variable compartida para los hilos niño, hilos monitores e hilo pintor.

servidor -> Instancia de ServerSocket que se utilizará como el socket del servidor.

Métodos:

escucharConsulta() -> en un bucle indefinido acepta conexiones con socket de clientes, y crea hilos de tipo Respuesta para atender a las correspondientes consultas.

run() -> ejecuta la acción principal de este hilo, lanzar el resto de hilos de la aplicación. Primeramente, un hilo que ejecuta el método escucharConsulta(), los monitores, y los niños, estos últimos los genera con intervalos entre ellos.

m. Escritor

Esta clase que hereda de hilo se encargará de escribir en el fichero log la evolución del campamento.

Atributos:

log -> Objeto de tipo PrintWriter utilizado para crear el fichero de texto en el que se mostrará la evolución del campamento.

colaMsg -> Cola concurrente que contiene los mensajes a escribir en el fichero.

Métodos:

addMsg(String msg) -> Método synchronized que se ocupa de añadir la fecha al mensaje a mandar y lo añade a la colaMsg.

print() -> Se ocupa de, en caso de haber mensajes, sacar el primero de la colaMsg y imprimirlo en el fichero log.

close() -> Cierra el fichero log.

run() -> Representa el ciclo de vida del escritor el cual se basa en indefinidamente llamar al método print.

n. Pintor

Esta clase que hereda de hilo se encargará de modificar la interfaz para que muestre en tiempo real el funcionamiento del campamento

Atributos:

campamento -> Instancia que representa la variable compartida de tipo Campamento.

JTextField -> Tendrá también como atributos todos los jTextField que

aparezcan en la interfaz principal.

Métodos:

pintar() -> Este método se encarga de poner en los jTextField los datos almacenados en sus respectivos contenedores. En el caso de la tirolina, dependiendo del estado en ese momento, se mostrará al niño únicamente en la posición correspondiente.

run() -> Representa el ciclo de vida del pintor el cual consistirá en indefinidamente llamar al método pintar intercalado por sleeps de 50 ms.

o. Respuesta

Esta clase que hereda de hilo se encargará de atender a las consultas generadas por clientes desde el lado del servidor.

Atributos:

numConsulta -> entero que identifica la consulta a responder.

ninno -> String que identifica al niño sobre el cuál se realice la consulta. Sólo en caso de ser necesario, hay consultas que no requieren introducir un niño.

conexion -> Socket que utilizará el cliente.

salida -> Stream de datos de salida para la conexión.

entrada -> Stream de datos de entrada para la conexión.

campamento -> instancia de la clase campamento de la cuál extraerá los datos para resolver la consulta.

Métodos:

run -> Se encarga de leer los datos del Stream de datos, y responder con la información contenida en el campamento dependiendo la consulta elegida.

p. Consulta

Esta clase que hereda de hilo se encargará de lanzar consultas al servidor para actualizar los datos en un cliente.

Atributos:

numConsulta -> entero el cual representa la consulta que se busca realizar.

ninno -> cadena que representa el id del niño del cual se busca saber cuántas actividades ha realizado.

textFieldCons -> JtextField de la consulta que se busca realizar.

carga -> JLabel que mostrará una C de que está cargando mientras espere a recibir una respuesta.

cliente -> Socket que se utilizará para realizar la conexión cliente-servidor.

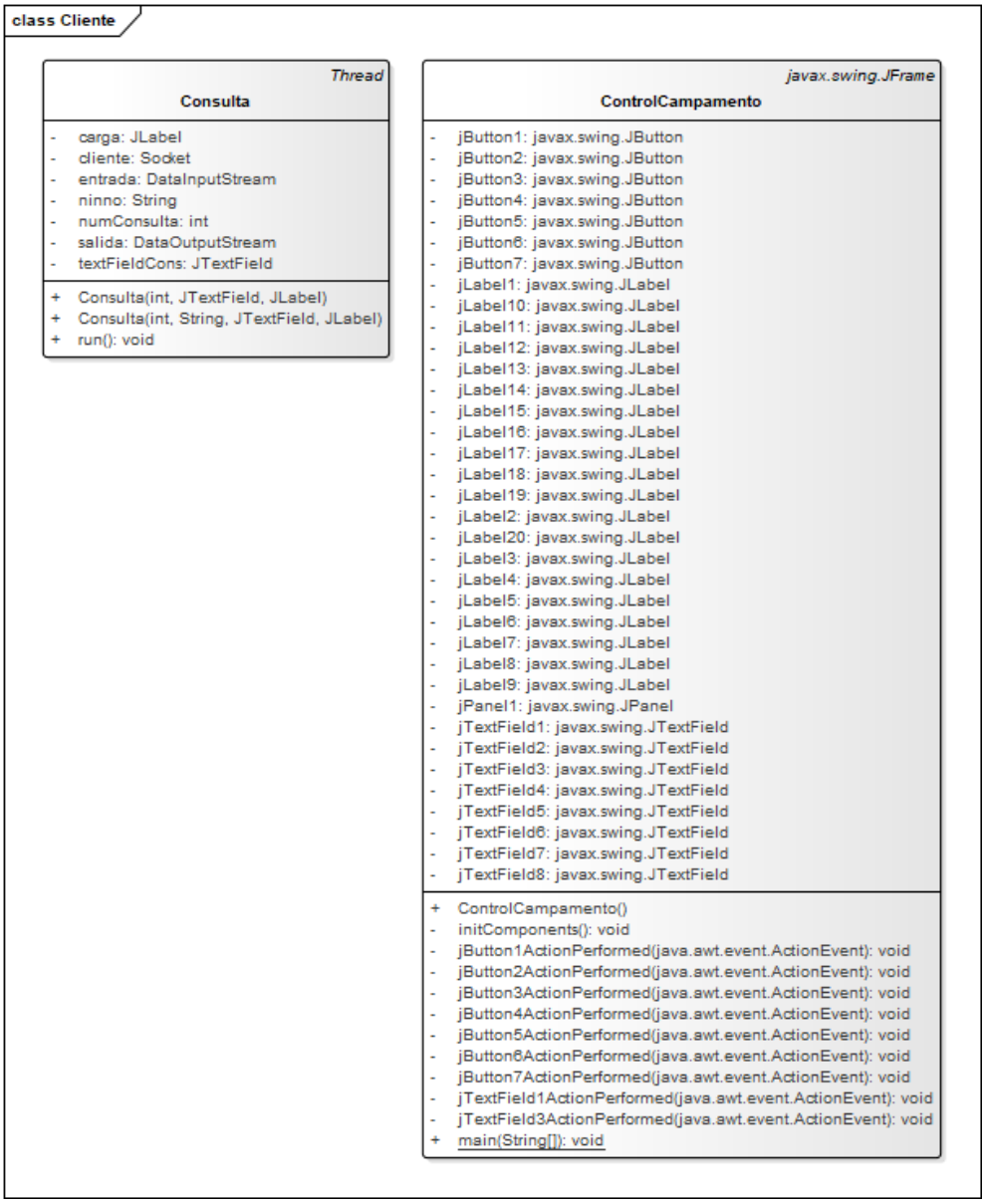
salida -> Stream de datos de salida para la conexión.

entrada -> Stream de datos de entrada para la conexión.

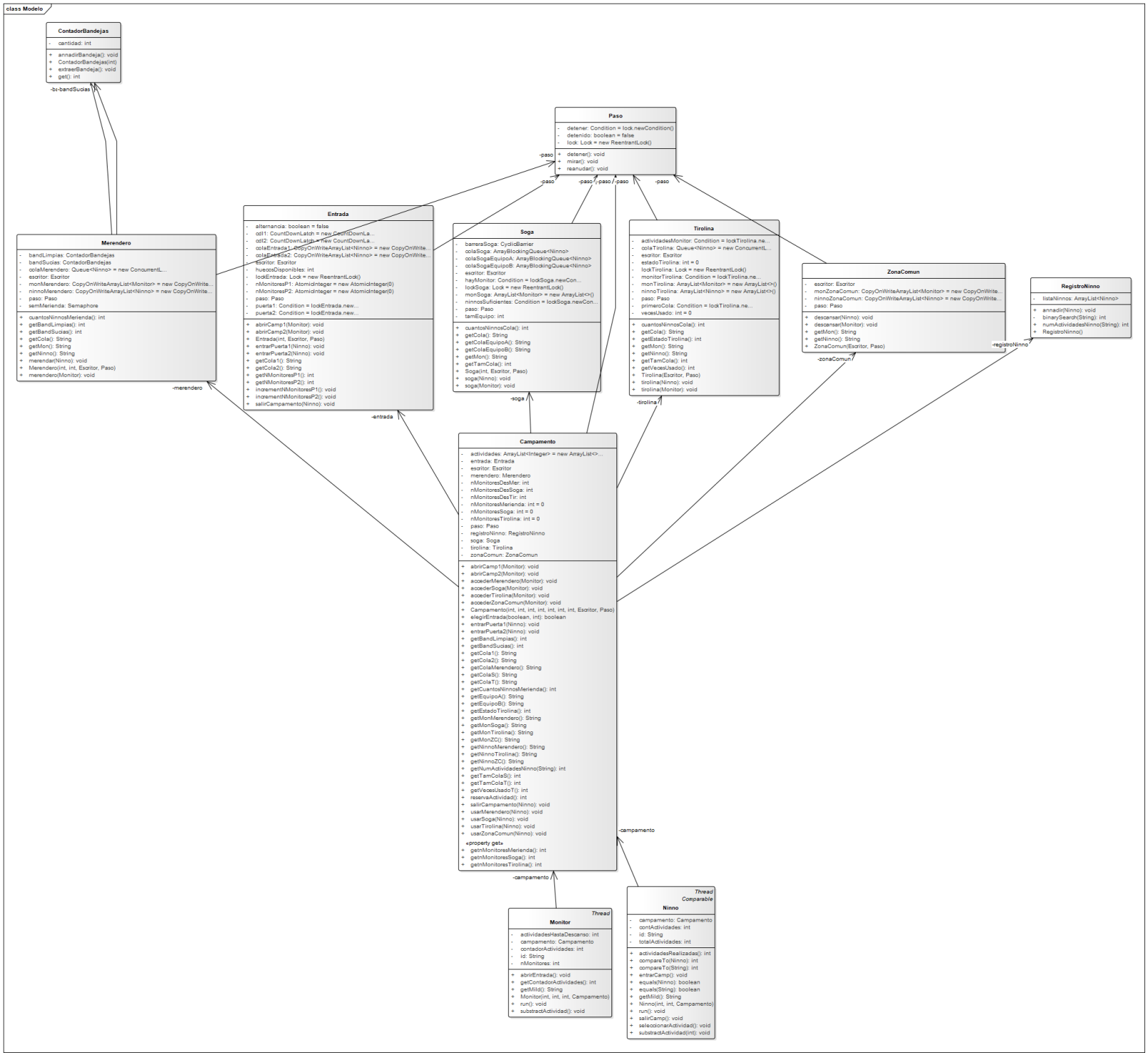
Métodos:

run() -> Envía los datos de la consulta por el Stream de datos, después espera a leer del Stream de datos, la respuesta a la consulta y la muestra por la interfaz. Mientras espera a la red, se muestra un icono de carga por la interfaz de usuario.

4. Diagrama de clases




class Interfaz
Thread
Pintor
- bandejasLimpias: JTextField
- bandejasSucias: JTextField
- campamento: Campamento
- colaEntrada1: JTextField
- colaEntrada2: JTextField
- colaMerendero: JTextField
- colaSoga: JTextField
- colaTirolina: JTextField
- dentroMerendero: JTextField
- dentroTirolina: JTextField
- equipoASoga: JTextField
- equipoBSoga: JTextField
- finTirolina: JTextField
- inicioTirolina: JTextField
- monitoresZC: JTextField
- monitorMerendero: JTextField
- monitorSoga: JTextField
- monitorTirolina: JTextField
- ninnosZC: JTextField
+ pintar(): void
+ Pintor(Campamento, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField, JTextField)
+ run(): void
-pintor
Thread
Escritor
- colaMsg: Queue<String>
- log: PrintWriter
+ addMsg(String): void
+ close(): void
+ Escritor()
+ print(): void
+ run(): void
-escritor
Thread
Generador
- campamento: Campamento
- conexion: Socket
- m: Monitor
- n: Ninno
- n_actividades_monitores: int
- n_actividades_ninnos: int
- n_monitores: int
- n_ninnos: int
- servidor: ServerSocket
+ escucharConsulta(): void
+ Generador(Campamento, int, int, int, int)
+ run(): void
-generador
Thread
Respuesta
- campamento: Campamento
- conexion: Socket
- entrada: DataInputStream
- ninno: String
- numConsulta: int
- salida: DataOutputStream
+ Respuesta(Socket, Campamento)
+ run(): void
javax.swing.JFrame
VentanaCampamento
- AFORO: int = 50
- AFORO_MERENDERO: int = 20
- campamento: Campamento
- escritor: Escritor
- generador: Generador
- jButton1: javax.swing.JButton
- jButton2: javax.swing.JButton
- jLabel1: javax.swing.JLabel
- jLabel10: javax.swing.JLabel
- jLabel11: javax.swing.JLabel
- jLabel12: javax.swing.JLabel
- jLabel13: javax.swing.JLabel
- jLabel14: javax.swing.JLabel
- jLabel15: javax.swing.JLabel
- jLabel16: javax.swing.JLabel
- jLabel17: javax.swing.JLabel
- jLabel18: javax.swing.JLabel
- jLabel19: javax.swing.JLabel
- jLabel2: javax.swing.JLabel
- jLabel20: javax.swing.JLabel
- jLabel21: javax.swing.JLabel
- jLabel22: javax.swing.JLabel
- jLabel23: javax.swing.JLabel
- jLabel24: javax.swing.JLabel
- jLabel25: javax.swing.JLabel
- jLabel3: javax.swing.JLabel
- jLabel4: javax.swing.JLabel
- jLabel5: javax.swing.JLabel
- jLabel6: javax.swing.JLabel
- jLabel7: javax.swing.JLabel
- jLabel8: javax.swing.JLabel
- jLabel9: javax.swing.JLabel
- jPanel1: javax.swing.JPanel
- jTextFieldColaMerendero: javax.swing.JTextField
- jTextFieldColaSoga: javax.swing.JTextField
- jTextFieldColaTirolina: javax.swing.JTextField
- jTextFieldEnTirolina: javax.swing.JTextField
- jTextFieldEntr1: javax.swing.JTextField
- jTextFieldEntr2: javax.swing.JTextField
- jTextFieldEquipoA: javax.swing.JTextField
- jTextFieldEquipoB: javax.swing.JTextField
- jTextFieldFinTirolina: javax.swing.JTextField
- jTextFieldLimpias: javax.swing.JTextField
- jTextFieldMonMerendero: javax.swing.JTextField
- jTextFieldMonSoga: javax.swing.JTextField
- jTextFieldMonTirolina: javax.swing.JTextField
- jTextFieldMonZonaComun: javax.swing.JTextField
- jTextFieldNinnoMerendero: javax.swing.JTextField
- jTextFieldNinnoZonaComun: javax.swing.JTextField
- jTextFieldPrepTirolina: javax.swing.JTextField
- jTextFieldSucias: javax.swing.JTextField
- MONITORES_MERIENDA: int = 2
- MONITORES_SOGA: int = 1
- MONITORES_TIROLINA: int = 1
- N_ACTIVIDADES_MONITORES: int = 10
- N_ACTIVIDADES_NINNOS: int = 15
- N_BANDEJAS: int = 25
- N_MONITORES: int = 4
- N_NINNOS: int = 20000
- paso: Paso
- pintor: Pintor
- TAM_EQUIPO: int = 5
- initComponents(): void
- jButton1ActionPerformed(java.awt.event.ActionEvent): void
- jButton2ActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldColaMerenderoActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldEnTirolinaActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldEquipoBActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldFinTirolinaActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldLimpiasActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldMonMerenderoActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldMonSogaActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldMonTirolinaActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldMonZonaComunActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldNinnoMerenderoActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldNinnoZonaComunActionPerformed(java.awt.event.ActionEvent): void
- jTextFieldPrepTirolinaActionPerformed(java.awt.event.ActionEvent): void
+ main(String[]): void
+ onExit(): void
+ VentanaCampamento()


class Modelo
ContadorBandejas
- cantidad: int
+ annadirBandeja(): void
+ ContadorBandejas(int)
+ extraerBandeja(): void
+ get(): int
-bt-bandSucias
Paso
- detener: Condition = lock.newCondition()
- detenido: boolean = false
- lock: Lock = new ReentrantLock()
+ detener(): void
+ mirar(): void
+ reanudar(): void
-paso
Entrada
- alternancia: boolean = false
- cd1: CountDownLatch = new CountDownLa...
- cd2: CountDownLatch = new CountDownLa...
- colaEntrada1: CopyOnWriteArrayList<Ninno> = new CopyOnWrite...
- colaEntrada2: CopyOnWriteArrayList<Ninno> = new CopyOnWrite...
- escritor: Escritor
- huecosDisponibles: int
- lockEntrada: Lock = new ReentrantLock()
- nMonitoresP1: AtomicInteger = new AtomicInteger(0)
- nMonitoresP2: AtomicInteger = new AtomicInteger(0)
- paso: Paso
- puerta1: Condition = lockEntrada.new...
- puerta2: Condition = lockEntrada.new...
+ abrirCamp1(Monitor): void
+ abrirCamp2(Monitor): void
+ Entrada(int, Escritor, Paso)
+ entrarPuerta1(Ninno): void
+ entrarPuerta2(Ninno): void
+ getCola1(): String
+ getCola2(): String
+ getNMonitoresP1(): int
+ getNMonitoresP2(): int
+ incrementNMonitoresP1(): void
+ incrementNMonitoresP2(): void
+ salirCampamento(Ninno): void
-entrada
Soga
- barreraSoga: CyclicBarrier
- colaSoga: ArrayBlockingQueue<Ninno>
- colaSogaEquipoA: ArrayBlockingQueue<Ninno>
- colaSogaEquipoB: ArrayBlockingQueue<Ninno>
- escritor: Escritor
- hayMonitor: Condition = lockSoga.newCon...
- lockSoga: Lock = new ReentrantLock()
- monSoga: ArrayList<Monitor> = new ArrayList<>()
- ninnosSuficientes: Condition = lockSoga.newCon...
- paso: Paso
- tamEquipo: int
+ cuantosNinnosCola(): int
+ getCola(): String
+ getColaEquipoA(): String
+ getColaEquipoB(): String
+ getMon(): String
+ getTamCola(): int
+ Soga(int, Escritor, Paso)
+ soga(Ninno): void
+ soga(Monitor): void
-soga
Tirolina
- actividadesMonitor: Condition = lockTirolina.ne...
- colaTirolina: Queue<Ninno> = new ConcurrentL...
- escritor: Escritor
- estadoTirolina: int = 0
- lockTirolina: Lock = new ReentrantLock()
- monitorTirolina: Condition = lockTirolina.ne...
- monTirolina: ArrayList<Monitor> = new ArrayList<>()
- ninnoTirolina: ArrayList<Ninno> = new ArrayList<>()
- paso: Paso
- primeroCola: Condition = lockTirolina.ne...
- vecesUsado: int = 0
+ cuantosNinnosCola(): int
+ getCola(): String
+ getEstadoTirolina(): int
+ getMon(): String
+ getNinno(): String
+ getTamCola(): int
+ getVecesUsado(): int
+ Tirolina(Escritor, Paso)
+ tirolina(Ninno): void
+ tirolina(Monitor): void
-tirolina
ZonaComun
- escritor: Escritor
- monZonaComun: CopyOnWriteArrayList<Monitor> = new CopyOnWrite...
- ninnoZonaComun: CopyOnWriteArrayList<Ninno> = new CopyOnWrite...
- paso: Paso
+ descansar(Ninno): void
+ descansar(Monitor): void
+ getMon(): String
+ getNinno(): String
+ ZonaComun(Escritor, Paso)
-zonaComun
RegistroNinno
- listaNinnos: ArrayList<Ninno>
+ annadir(Ninno): void
- binarySearch(String): int
+ numActividadesNinno(String): int
+ RegistroNinno()
-registroNinno
Merendero
- bandLimpias: ContadorBandejas
- bandSucias: ContadorBandejas
- colaMerendero: Queue<Ninno> = new ConcurrentL...
- escritor: Escritor
- monMerendero: CopyOnWriteArrayList<Monitor> = new CopyOnWrite...
- ninnoMerendero: CopyOnWriteArrayList<Ninno> = new CopyOnWrite...
- paso: Paso
- semMerienda: Semaphore
+ cuantosNinnosMerienda(): int
+ getBandLimpias(): int
+ getBandSucias(): int
+ getCola(): String
+ getMon(): String
+ getNinno(): String
+ merendar(Ninno): void
+ Merendero(int, int, Escritor, Paso)
+ merendero(Monitor): void
-merendero
Campamento
- actividades: ArrayList<Integer> = new ArrayList<>...
- entrada: Entrada
- escritor: Escritor
- merendero: Merendero
- nMonitoresDesMer: int
- nMonitoresDesSoga: int
- nMonitoresDesTir: int
- nMonitoresMerienda: int = 0
- nMonitoresSoga: int = 0
- nMonitoresTirolina: int = 0
- paso: Paso
- registroNinno: RegistroNinno
- soga: Soga
- tirolina: Tirolina
- zonaComun: ZonaComun
+ abrirCamp1(Monitor): void
+ abrirCamp2(Monitor): void
+ accederMerendero(Monitor): void
+ accederSoga(Monitor): void
+ accederTirolina(Monitor): void
+ accederZonaComun(Monitor): void
+ Campamento(int, int, int, int, int, int, int, int, Escritor, Paso)
+ elegirEntrada(boolean, int): boolean
+ entrarPuerta1(Ninno): void
+ entrarPuerta2(Ninno): void
+ getBandLimpias(): int
+ getBandSucias(): int
+ getCola1(): String
+ getCola2(): String
+ getColaMerendero(): String
+ getColaS(): String
+ getColaT(): String
+ getCuantosNinnosMerienda(): int
+ getEquipoA(): String
+ getEquipoB(): String
+ getEstadoTirolina(): int
+ getMonMerendero(): String
+ getMonSoga(): String
+ getMonTirolina(): String
+ getMonZC(): String
+ getNinnoMerendero(): String
+ getNinnoTirolina(): String
+ getNinnoZC(): String
+ getNumActividadesNinno(String): int
+ getTamColaS(): int
+ getTamColaT(): int
+ getVecesUsadoT(): int
+ reservaActividad(): int
+ salirCampamento(Ninno): void
+ usarMerendero(Ninno): void
+ usarSoga(Ninno): void
+ usarTirolina(Ninno): void
+ usarZonaComun(Ninno): void
«property get»
+ getnMonitoresMerienda(): int
+ getnMonitoresSoga(): int
+ getnMonitoresTirolina(): int
-campamento
Thread
Monitor
- actividadesHastaDescanso: int
- campamento: Campamento
- contadorActividades: int
- id: String
- nMonitores: int
+ abrirEntrada(): void
+ getContadorActividades(): int
+ getMiId(): String
+ Monitor(int, int, int, Campamento)
+ run(): void
+ substractActividad(): void
Thread
Comparable
Ninno
- campamento: Campamento
- contActividades: int
- id: String
- totalActividades: int
+ actividadesRealizadas(): int
+ compareTo(Ninno): int
+ compareTo(String): int
+ entrarCamp(): void
+ equals(Ninno): boolean
+ equals(String): boolean
+ getMiId(): String
+ Ninno(int, int, Campamento)
+ run(): void
+ salirCamp(): void
+ seleccionarActividad(): void
+ substractActividad(int): void

5. Código fuente