**GRADO EN INGENIERÍA DE COMPUTADORES**

**CURSO 2024-2025 CONVOCATORIA ORDINARIA**

**03223193L – MARTÍNEZ MARTÍN, ALEJANDRO**

**09147371H – CABRERA ROZALÉN, RAÚL**

ÍNDICE

Contenido

[1. ANÁLISIS DE ALTO NIVEL 3](#_Toc197261104)

[1.1. DESCRIPCIÓN GENERAL DEL PROBLEMA 3](#_Toc197261105)

[1.2. ACTORES PRINCIPALES 3](#_Toc197261106)

[1.2.1. Humanos 3](#_Toc197261107)

[1.2.2. Zombis 3](#_Toc197261108)

[1.2.3. Zonas del refugio. 4](#_Toc197261109)

[1.2.4. Zonas de riesgo 4](#_Toc197261110)

[1.2.5. Túneles 4](#_Toc197261111)

[2. DISEÑO GENERAL DEL SISTEMA Y DISCUSIÓN DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS 5](#_Toc197261112)

[2.1. DISEÑO GENERAL DEL SISTEMA 5](#_Toc197261113)

[2.2. DISCUSIÓN DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS 9](#_Toc197261114)

[3. DESCRIPCIÓN DE LAS CLASES PRINCIPALES 10](#_Toc197261115)

[4. DIAGRAMA DE CLASES 10](#_Toc197261116)

[5. CÓDIGO FUENTE 11](#_Toc197261117)

# ANÁLISIS DE ALTO NIVEL

## DESCRIPCIÓN GENERAL DEL PROBLEMA

El enunciado nos pide generar un sistema concurrente que simule un escenario en el que la humanidad ha sufrido un apocalipsis zombi mediante hilos en Java. En este mundo apocalíptico los humanos viven en un refugio seguro donde los zombis nunca pueden entrar, pero necesitan comer, por lo que van al exterior a través de cuatro túneles que conectan el refugio con cuatro zonas de riesgo distintas. En estas zonas de riesgo, los humanos tratan de recolectar comida para posteriormente llevarla al refugio, pero pueden ser atacados por zombis, lo que conllevará a que regresen al refugio sin comida o a que mueran y se conviertan en zombis. Dentro del refugio hay una zona común donde los humanos se preparan para ir al exterior, un comedor donde comen y depositan la comida al volver del exterior, y una zona de descanso donde descansan al volver del exterior o para recuperarse del ataque de un zombi.

Se debe gestionar el sistema de manera concurrente en diferentes zonas, como en el acceso a los túneles y la manera en la que los zombis interactúan con los humanos. Para ello se utilizará programación concurrente en Java.

* Túneles: Los túneles solo permiten el paso de un humano a la vez entre el exterior y el refugio. Además, los humanos que quieran volver al refugio desde el exterior tendrán prioridad frente a los que quieran salir del mismo.
* Zombis: Los zombis solo pueden atacar a un humano a la vez, y un humano solo puede ser atacado por un zombi al mismo tiempo.

Finalmente, se debe permitir un acceso remoto desde un cliente al servidor, que contiene toda la lógica del programa. Este cliente recibirá información en tiempo real sobre los humanos y zombis en las diferentes zonas, así como el contador de muertes de los zombis. Además, se deben generar interfaces gráficas que muestren esta información de manera interactiva.

## ACTORES PRINCIPALES

### Humanos

Son los personajes que deben sobrevivir en el refugio y a los ataques de los zombis cuando salen en busca de comida para llevar al refugio. Los humanos interactúan entre sí en los túneles e interactúan con los zombis en las zonas de riesgo.

### Zombis

Son los personajes que tratan de acabar con la humanidad. Se mueven por las zonas de riesgo aleatoriamente en busca de humanos para atacarlos. Inicialmente solo hay uno, pero a medida que van muriendo humanos van creándose más zombis. Un humano muere cuando es atacado por un zombi y el ataque resulta exitoso, lo cual ocurre con una probabilidad de 1/3.

### Zonas del refugio.

Aunque las zonas no son actores como tal, son esenciales para las interacciones entre los humanos, al igual que las zonas de riesgo y los túneles.

El refugio se divide en 3 zonas:

* Zona común: zona en la que los humanos se preparan y seleccionan el túnel por el que accederán a la zona de riesgo.
* Zona de descanso: zona en la que los humanos descansan según vuelven del exterior y donde, si han sido atacados pero han sobrevivido, descansarán por segunda vez después de comer para curarse del ataque del zombi.
* Comedor: zona en la que los humanos depositan la comida al volver del exterior si no han sido atacados y donde comen tras descansar antes de volver a la zona común y prepararse para volver a salir.

### Zonas de riesgo

Las zonas de riesgo están fuera del refugio y es donde interactúan los humanos y los zombis. Los humanos acceden a ella en busca de comida, pero pueden ser atacados si hay zombis en esa zona.

Si son atacados y no mueren, regresan al refugio marcados, lo que significa que tiene que descansar dos veces, y sin comida.

En el caso en el que un zombi ataque de manera exitosa a un humano, este morirá y se convertirá en zombi.

### Túneles

Son los medios mediante los que los humanos acceden al refugio y salen de este. Solo un humano puede pasar por cada túnel a la vez, y se necesita un grupo de tres humanos listos para salir por ese túnel para que puedan salir. Para volver esto no es necesario, puede volver únicamente un humano. Un humano que quiere volver al refugio tiene prioridad sobre los humanos que quieren salir del refugio.

# DISEÑO GENERAL DEL SISTEMA Y DISCUSIÓN DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS

## DISEÑO GENERAL DEL SISTEMA

El diseño del sistema está basado en programación concurrente, en la que se utilizan hilos (threads) para modelar las entidades del sistema y sus interacciones.

Componentes:

1. Humanos (Clase Humano.java):

Los humanos son hilos que realizan diferentes tareas en las distintas zonas que existen. Cada humano sigue un ciclo de vida definido:

1. Preparación: los humanos inician en la zona común del refugio y se preparan para salir.
2. Salida del refugio: después de prepararse, el humano elige uno de los cuatro túneles para dirigirse a una zona de riesgo. En ese túnel espera a que sean tres para comenzar a acceder al túnel y salir del refugio.
3. Regreso al refugio: una vez hayan terminado su estancia en la zona de riesgo, ya sea porque han recolectado la comida sin problema o porque han sido atacados sin éxito por un zombi, vuelven al túnel marcados o sin marcar y con o sin comida, y regresan al refugio con prioridad frente a los humanos que quieren salir.
4. Recuperación en el refugio: una vez hayan regresado, lo primero que hacen es depositar la comida las dos unidades de comida si han conseguido volver con ella, después se dirigen a la zona de descanso y descansan, al terminar se dirigen al comedor y comen una pieza de comida, y a continuación, si han sido atacados y están marcados, descansan por segunda vez para recuperarse del ataque. Lo próximo que hacen es volver a la zona común y repetir el ciclo.

Cada uno de estos comportamientos son gestionados por hilos independientes, lo que permite que varios humanos interactúen simultáneamente con el sistema.

1. Zombis (Clase Zombie.java)

Los zombis también son hilos que van alternando entre las zonas de riesgo de manera aleatoria en busca de humanos para atacarlos.

Inicialmente hay un solo zombi que buscará humanos para matarlos y convertirlos también en zombis.

Comportamiento de los zombis:

1. Deambulación: Los zombis se mueven aleatoriamente entre las zonas de riesgo. Cuando acceden a una y no hay humanos, esperan entre dos y tres segundos y cambian de zona.
2. Ataque: encuentran a uno o varios humanos, eligen a uno y lo atacan. El ataque tiene una probabilidad de éxito de una entre tres. Si el ataque resulta exitoso, el humano se convierte en zombi, y si no, el humano vuelve al refugio.
3. Cambio de zona: Ya sea porque no han encontrado humanos en la zona de riesgo en la que está o porque ha atacado a un humano, el zombi espera entre dos y tres segundos y cambian aleatoriamente de zona.
4. Túneles (Clase Tunel.java)

Son conexiones entre el refugio y las zonas de riesgo. Los túneles se pueden dividir principalmente en 3 zonas:

* + Zona de espera: los humanos antes de acceder al túnel desde el refugio tienen que esperar para formar un grupo de tres humanos.
  + Zona de paso: Una vez que se hayan formado el grupo, de uno en uno y fijándose siempre en que no haya humanos queriendo volver, los humanos del grupo previamente formado entran al túnel y acceden a la zona de riesgo correspondiente.
  + Zona de espera para volver: Cuando los humanos terminan su estancia en la zona de riesgo, estos vuelven por el mismo túnel por el que salieron con prioridad frente a los humanos que quieren salir.

Por lo tanto, las características del túnel son:

1. Acceso exclusivo al túnel, donde solo un humano puede pasar por el túnel a la vez.
2. Prioridad para regresar de los humanos que se encuentran fuera del refugio con respecto a los que están dentro.

1. Zonas del refugio (Clases Comedor.java, ZonaDescanso.java y ZonaComun.java)

Dentro del refugio hay diferentes zonas donde los humanos realizan diferentes actividades. Estas zonas se gestionan de tal forma para que los humanos no interfieran entre sí.

1. Comedor: es una zona del refugio donde los humanos depositan la comida que traen del exterior, siempre que no sean atacados, y donde comen. Cuando un humano regresa con comida, deposita dos piezas en el comedor. Por otro lado, cuando un humano come, lo hace siempre en orden de llegada, y si no hay comida, los humanos esperarán igualmente en el orden en el que han llegado hasta que algún humano deposite comida en el comedor para poder comer. Este proceso de alimentación debe gestionarse de manera sincronizada, ya que varios humanos pueden intentar acceder al comedor al mismo tiempo, y el número de piezas de comida es limitado.

Funcionamiento del comedor:

1. Acceso ordenado a la comida.
2. Sincronización en el acceso al comedor para garantizar que los humanos no interfieran entre sí al acceder a la comida.
3. Zona de descanso: es una zona del refugio donde los humanos van a descansar antes de comer al volver del exterior, y para curarse antes de comenzar de nuevo su ciclo si han sido atacados por un zombi en las zonas de riesgo.
4. Zona común: En esta zona del refugio los humanos se preparan para ir al exterior. Aleatoriamente eligen un túnel por el que van a salir y esperan a que tres humanos, incluido él, quieran salir por ese túnel hacia la zona de riesgo correspondiente. Es una zona de preparación donde no se realiza nada más a parte de la selección del túnel y la formación de grupos para salir por el túnel.
5. Zonas de riesgo (Clase ZonaRiesgo.java)

Son áreas externas al refugio a las que se accede mediante los túneles donde los humanos intentan permanecer entre tres y cinco segundos sin ser atacados por un zombi para recolectar dos piezas de comida. En el caso de que concluya el tiempo y no hayan sido atacados, vuelve al refugio y depositan la comida recolectada en el comedor. Si son atacados por un zombi mientras intentan recolectar comida hay dos opciones:

1. El ataque del zombi falla: Si esto ocurre vuelven inmediatamente al refugio marcados y soltando la comida. Después de comer deberán volver a descansar para dejar de estar marcados.
2. El ataque del zombi resulta exitoso: El humano muere y se crea un zombi con el mismo identificador del humano, cambiando la H de humano por la Z de zombi.
3. Arranque (Clase Arranque.java)

La clase arranque es la encargada de iniciar la simulación. Dentro de esta clase se crea el zombi inicial y se crean los humanos, así como las zonas que componen el escenario apocalíptico. La clase crea constantemente humanos de manera ordenada cada 0,5-2 segundos, asignándoles un identificador a cada uno de manera ordenada. En esta clase también se interactúa con la clase pausa, que se encarga de pasar la actividad de los hilos (zombis y humanos).

Por último, la información que genera la simulación se muestra de manera gráfica en una ventana (VentanaServ).

1. Pausa (Clase Pausa.java)

Esta clase se encarga de pausar y reanudar la ejecución del sistema, y además tiene un método para que los hilos comprueben si se ha pausado o no la simulación para que puedan detenerse.

1. Ranking (Clase Ranking.java)

Esta clase se encarga de conocer todos los contadores de muertes de los zombis que haya en el sistema y, a partir de ellos, crear un ranking de los tres zombis con más muertes causadas a los humanos.

1. Servidor (Clase Servidor.java)

El servidor se encarga de crear un servidor donde se inicie una simulación del apocalipsis. Este servidor intercambiará información con el cliente, donde envía datos de los humanos y zombis que hay en cada zona del sistema, así como los datos de las muertes causadas por los tres mejores zombis. También se encarga de recibir las ordenes de los clientes para parar y reanudar la simulación.

1. Cliente (Clase Cliente.java)

El cliente recibe información sobre la cantidad de humanos y zombis que hay en cada zona. También recibe información sobre el ranking de zombis en cuanto a las muertes que han causado. Con esta información se crea una ventana donde se muestran gráficamente los resultados obtenidos del servidor (VetanaCli).

1. Ventanas (Clases VentanaServ.java y VentanaCli.java)

Son ventanas gráficas que muestran información sobre los humanos y zombis que hay en cada zona, así como las muertes causadas por los tres zombis más letales en el caso de la ventana del cliente (VentanaCli).

1. Logger (Clase Logger.java)

Esta clase se encarga de registrar los eventos que van sucediendo en el sistema, como los accesos de los humanos a diferentes zonas, los depósitos de comida, las acciones que realizan los humanos y las interacciones que tienen con los zombis. Todos estos sucesos se escriben en un documento de texto llamado Apocalipsis.txt. Esta clase es crucial para conocer el comportamiento del sistema.

## DISCUSIÓN DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS

**Clase Tunel.java**   
Herramientas empleadas:

* Semaphore(3, true) esperarAntes: Controla el acceso a la zona de espera para formar grupos de tres humanos. El parámetro true asegura que el semáforo sea FIFO.
* CyclicBarrier(3) esperar: Asegura que los humanos solo accedan al túnel cuando el grupo de tres esté formado.
* Semaphore pasar: Gestiona el acceso al túnel de uno en uno, asegurando que solo un humano pase a la vez.
* LinkedBlockingQueue<Humano> listaPasar, listaPasado y listaRegresar: Utilizada para gestionar las listas de espera y de acceso al túnel de los humanos, garantizando un acceso ordenado y limitado.
* ReentrantLock con condition: Verifica que no haya humanos intentando regresar antes de permitir que otros salgan, gestionando el acceso con prioridad.

**Clase Comedor.java**   
Herramientas empleadas:

* AtomicInteger comidaDisponible: permite un acceso atómico a la comida disponible, asegurando operaciones seguras y sin interferencias.
* Semaphore (1,true) comer: Controla que solo un humano pueda comerá ala vez, mientras los demás esperan en orden.
* ReentrantLock con condition comidaEsperar y noComida: gestiona la espera de los humanos cuando no hay comida disponible.
* LinkedBlockingQueue<Humano> listaComedor: organiza la lista de los humanos en el comedor, gestionando su acceso y salida de forma eficiente y ordenada.

**Clase Humano.java   
Herramientas empleadas:**

* **CyclicBarrier(2) esperarAtaque: Se utiliza para coordinar que el humano y el zombi se sincronicen durante el ataque. La barrera se activa solamente cuando el humano y el zombi están listos,. Esto asegura que el ataque no ocurra hasta que ambos estén preparados. Espera a que los dos estén listos para continuar, ya que es un CyclicBarrier de 2.**
* **AtomicBoolean esperandoAtaque: Se usa para indicar de manera atómica si el humano está esperando un ataque de un zombi. Si este está en true significa que puede ser atacado por un zombi, mientras que si está en false significa que ya está siendo atacado por un zombi, para que no ataquen dos zombis al mismo humano.**

**Clase Logger.java**Herramientas empleadas:

* Semaphore(1, true) sem: controla el acceso al archivo de log, para asegurar que solo un hilo escriba a la vez en el y no haya interferencias. Al ser true significa que es un semáforo FIFO y que se accede y se espera en orden de llegada.

**Clase Pausa.java**   
Herramientas empleadas:

* ReentrantLock lockDetener y condition condición: se usan para gestionar la espera y la reanudación de los hilos, permitiendo que se bloqueen y desbloqueen de manera sincronizada según indique la simulación.

**Clase ZonaComun.java**   
Herramientas empleadas:

* LinkedBlockingQueue<Humano> listaHumanos: se utiliza para obtener y modificar la lista de los humanos que están en la zona común del refugio de manera eficiente y ordenada.

**Clase ZonaDescanso.java**   
Herramientas empleadas:

* LinkedBlockingQueue<Humano> listaDescansando: se utiliza para obtener y modificar la lista de los humanos que están descansando en la zona de descanso del refugio de manera eficiente y ordenada.

**Clase ZonaRiesgo.java**   
Herramientas empleadas:

* LinkedBlockingQueue<Humano> listaZombies: se utiliza para obtener y modificar la lista de los zombis que están en la zona de riesgo indicada de manera eficiente y ordenada.

# DESCRIPCIÓN DE LAS CLASES PRINCIPALES

Para no repetir el 2.1, se describirán los métodos principales de las clases

## Parte1

**Clase Sevidor.java**iniciarServ()

Se crea un socket para el servidor y se llama al arranque para que empiece la ejecución del programa. Una vez el cliente acepta la conexión se llama al método conexionCliente(Socket cli)

conexionCliente(Socket cli)

Cuando este método se llama se crean los I/O stream para mandar y recibir los Objetos. Se manda toda la información de la ejecución y se recibe del cliente un Boolean por si se quisiera parar la ejecución.

**Clase Arranque.java**crearSimulacionSegundoPlano()

Lanza un hilo llamando a iniciarSimulacion() para que la clase Arranque no se quede bloqueada instanciando los miles de hilos.

iniciarSimulacion()

Aquí es donde se instancias las principales clases del proyecto, tanto los arrays de los túneles como los Humanos y Zombies

pausarEjecucion()

Este método es llamado para hacer uso de la clase Pausa y que pare la ejecución.

reanudarEjecucion()

Realiza una función similar a pausarEjecucion(), pero esta vez la clase Pausa reanuda la ejecución.

Los métodos getters() de las instancias que hemos creado de las clases principales.

**Clase Humano.java**run()

El bucle while en donde el humano realiza sus tareas y comprueba si tiene que parar llamando a la clase Pausa.

defensa()

Hace uso de un CyclicBarrier para esperarse a que el Zombie termine de atacarle.

comprobarPausaHumano()

Se utiliza en Clases ajenas para que el humano compruebe si tiene que pausarse.

**Clase Zombie.java**run()

Aquí es donde se encuentra el while en el que reside la lógica del Zombie. Comprueba si hay que pausarse, elige aleatoriamente una zona de Riesgo y escoge un humano para atacarlo.

atacar()

Si el humano ha tenido la mala suerte de morir, se crea un nuevo zombie con su mismo ID y se llama a matarHumano(). En el caso contrario se marca al humano y se le deja ir.

matarHumano()

Se le cambia la condición al humano de No Muerto a Muerto mediante una llamada a la función morir() del humano, esta función pone a true el booleano de muerto.

**Clase Tunel.java**irExterior(Humano hu)

Aquí se encuentra la lógica del funcionamiento de los túneles, donde se esperan los humanos y van pasando de uno en uno de forma ordenada.

venirDelExterior(Humano hu)

Este método se encarga de que los humanos vuelvan por el túnel y tengan prioridad sobre los que van a pasar a las zonas de riesgo.

**Clase Comedor.java**depositarComida(Humano hu)

Se comprueba si el humano ha sido marcado o no y en el caso de que no esté marcado depositará 2 unidades de comida.

comer(Humano hu)

El humano tomará una pieza de comida y si no queda más comida esperan de manera ordenada a que otro superviviente la traiga.

**Clase ZonaRiesgo.java**devolverHumanoAleatorio(ZonaRiesgo zonaActual)

Este método se encarga de que los zombies cojan aleatoriamente a un Humano de manera que solamente un Zombie pueda coger a un humano en concreto

**Clase Pausa.java**pararEjecucion()

Se cambia el valor de booleano “parar” a true y los hilos se van esperando en la cola de la Condition que tiene esta clase cuando comprueban el booleano.

continuarEjecucion()

Se cambia el valor del booleano a false y se realiza un signalAll para despertar a todos los hilos que están a la espera.

comprobarPausa()

Los hilos comprueban si hay que detener la ejecución del programa y si es así realizan un await() para esperar.

**Clase Ranking.java**hacerRanking()

Se crea una lista que incluye a todos los zombies de todas las zonas de riesgo y se ordena de manera que los tres primeros sean los que más muertes llevan.

**Clase Logger.java**escribir(String msg)

Haciendo uso del semáforo, los hilos escriben el archivo apocalipsis.txt todo lo que va sucediendo en el programa

**Clase ZonaComun.java**entrarZonaComun(Humano hu)

Se añaden a la lista de la zona común y actualizan la ventana del servidor.

prepararse(Humano hu)

Aquí se preparan ejecutando un sleep de entre 1 o 2 segundos.

vidaFueraRefugio(Humano hu)

Aquí es donde el humano elige aleatoriamente el túnel por el que va a pasar y se elimina de la lista de la zona común.

**Clase ZonaDescanso.java**

descansarVuelta(Humano hu)

El humano descansa tras su vuelta descansará entre 2 y 4 segundos registrándolo en el log.

descansarMarcado(Humano hu)

Si el humano ha sido marcado ejecutará este método en el que volverá a la zona de descanso para dormir entre 3 y 5 segundos.

**Clase VentanaServ.java**

Métodos de actualización

A esta clase se le pasa como parámetro la clase Arranque, de la que se sacan todo el resto de clases con métodos get() para posteriormente obtener las listas pertinentes de donde sacar la información con otro get().

## Parte2

**Clase Cliente.java**

conectarServ()

En este método reside la lógica con la que se comunica el cliente con el servidor. Se crea el socket para conectarse en el puerto correspondiente y los I/O streams para recibir y mandar objetos. Se reciben los datos que muestra la pantalla del cliente y se manda un Boolean para indicar si se pausa o no la ejecución.

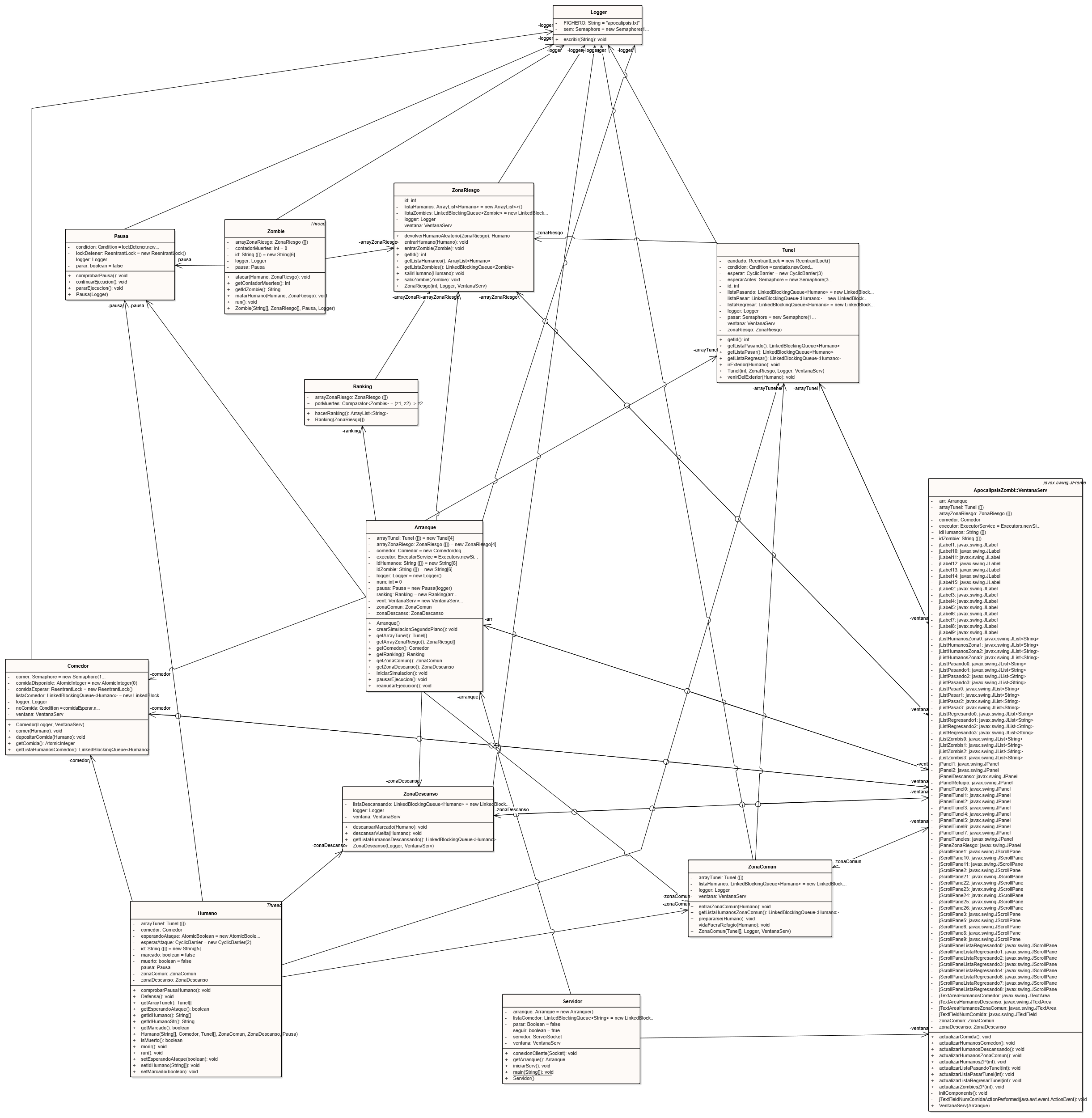
Métodos get() de los atributos recibidos del servidor.

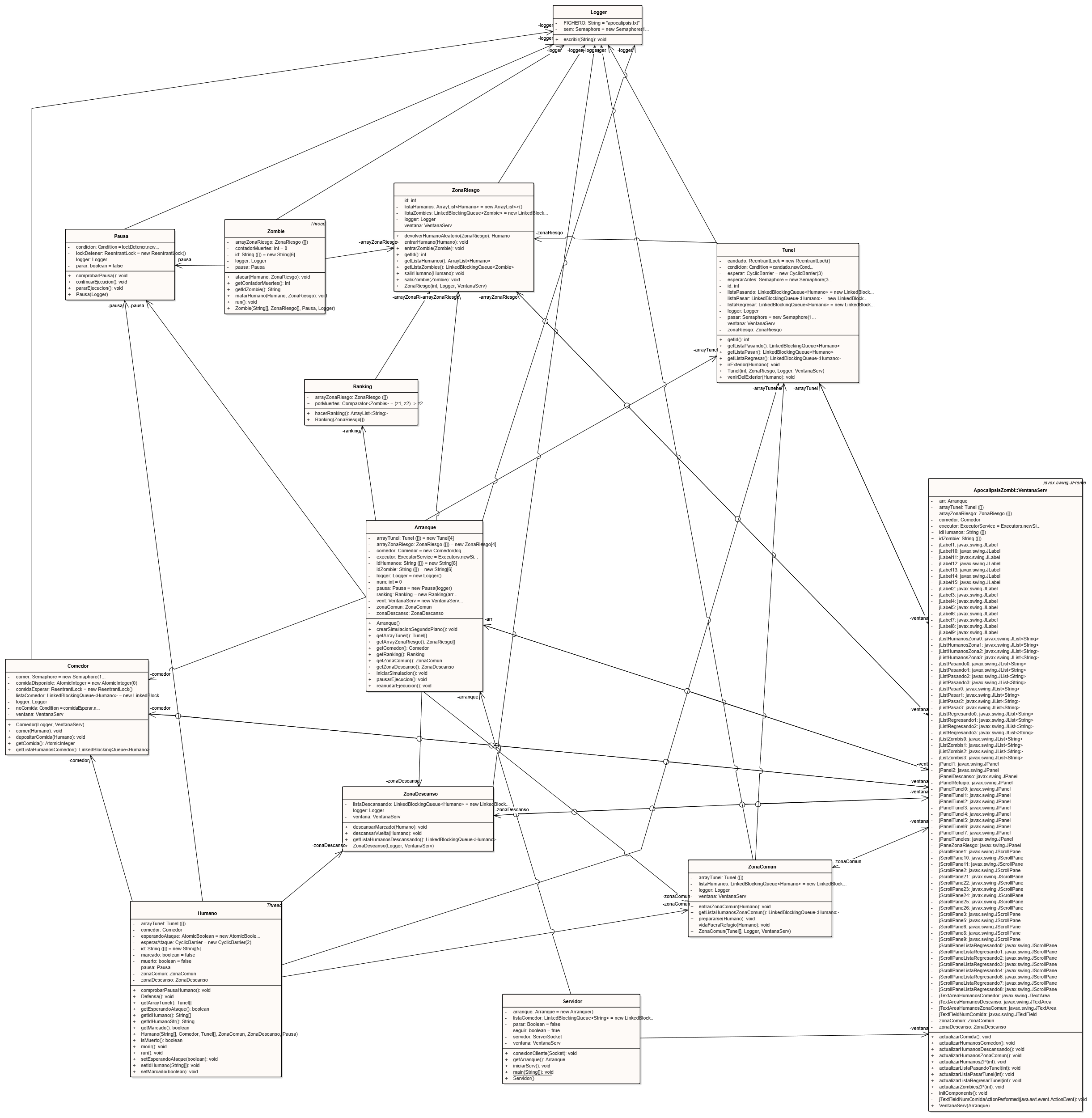
**Clase VentanaCli.java**

descansarVuelta(Humano hu)

Para mostrar los datos se pasa la clase Cliente por el constructor y se realizan get() de los valores que se reciben del servidor.

# DIAGRAMA DE CLASES





# CÓDIGO FUENTE