Programación Concurrente

Master de Computación
Facultad de Ciencias
Universidad de Cantabria.

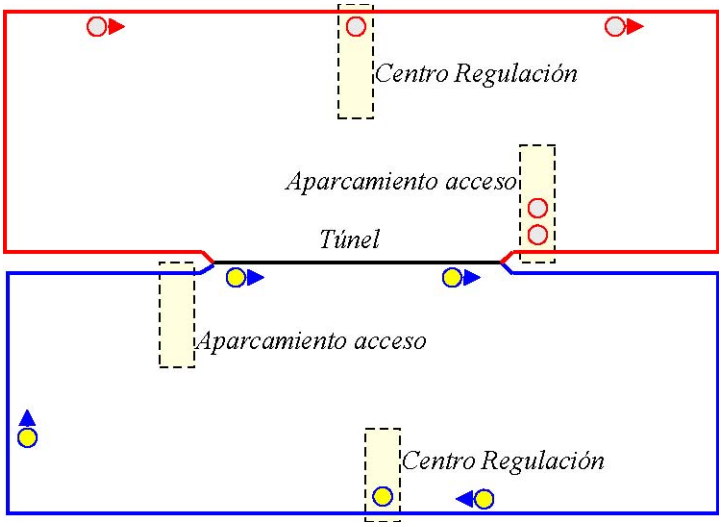
Documento: Versión básica (secuencial) de la aplicación RedFerroviaria

Autores: José M. Drake
Laura Barros

Fecha: 20-10-08

Aplicación RedFerroviaria

La ***red ferroviaria*** se compone de un circuito con dos ***tramos ferroviarios*** independientes y cerrados que son recorridos por un conjunto de **trenes**. En cada tramo los trenes circulan en la misma dirección y a la misma velocidad.



Cada tramo ferroviario tiene un ***centro de regulación***, en el que los trenes se aparcan temporalmente a fin de regular su circulación para que su tráfico sea uniforme. Cuando un tren llega al centro de regulación, no se le permite salir antes de que haya transcurrido un tiempo igual tiempo que tarda un tren en recorrer su tramo partido por el número de trenes en el tramo.

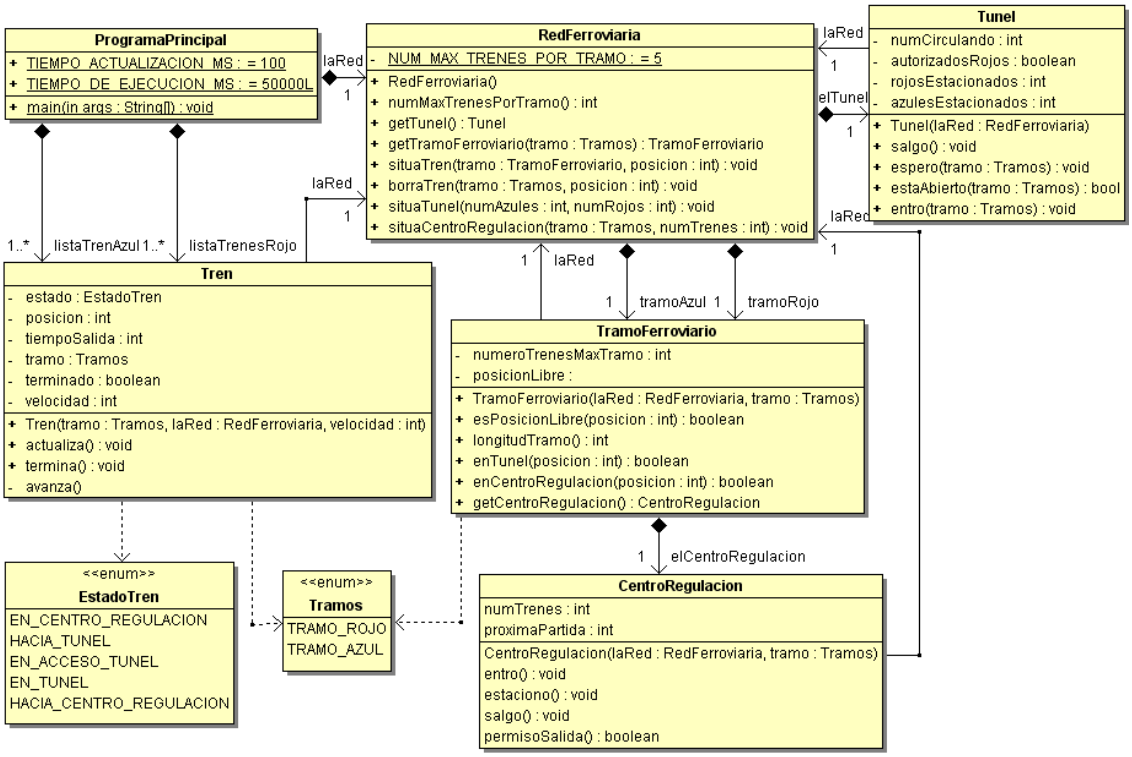
La red ferroviaria contiene un túnel que es compartido por los trenes de ambos tramos ferroviarios. En el túnel sólo hay una vía que es utilizada por los trenes de ambos tramos ferroviarios. Los trenes de cada tramo ferroviario pasan por el túnel en sentido contrario, por lo que mientras que un tren de un tramo ferroviario esté atravesando el túnel, los trenes que lleguen del otro tramo ferroviario deben estacionarse en el acceso del túnel hasta que no haya trenes circulando en sentido contrario. Sin embargo dos trenes que circulen en el mismo sentido, si pueden circular simultáneamente por el túnel.

La red ferroviaria representa la infraestructura de señalización, y de ella, los trenes pueden obtener información sobre su situación respecto de los elementos de la vía, del tiempo, y enviar a los recursos aviso de su situación y estado.

Diseño de la aplicación

Criterios de diseño

Como se muestra en el diagrama de clases de la figura, la aplicación se ha construido con cinco clases que representan los principales tipos de objetos que intervienen en la aplicación: RedFerroviaria, TramoFerroviario, Tren, Tunel y CentroRegulación.



Criterios de diseño estructural:

- Se ha establecido una estructura de contenedores de tipo árbol definido por las relaciones de composición. La raíz la es RedFerroviaria, el TramoFerroviario es un contendor intermedio, y las clases Trenes, Tunel y CentroRegulación son los elementos terminales u hojas.
- Desde todos los elementos, se proporciona acceso a la clase raíz RedFerroviaria, y a partir de ella, y haciendo uso de la estructura en árbol de los componentes se puede acceder a cualquier otro, sin necesitar pasar todas los accesos a través de los constructores.
- Se ha buscado que no haya redundancia de información. Cada dato sólo es mantenido en el objeto que lo genera o en el que por su naturaleza es responsable de él. Si un objeto requiere un dato de forma repetida, lo puede almacenar localmente de forma privada y es responsable de que sea coherente con el valor oficial. Para evitar transmitir fallos de coherencia, todos deben buscar la información en el objeto responsable de ella.

Criterios dinámicos:

La aplicación es de naturaleza típicamente concurrente. Los trenes avanza interaccionando con los elementos que encuentran, y decidiendo continuar o parar según el estado el del elemento (Tunel o CentroRegulación) que encuentra.

Descripción de las clases

Clase **RedFerroviaria**: Representa a la red ferroviaria completa. Es el contenedor raíz de los restantes elementos, y a través de ella se puede localizar cualquier elemento. Constituye la interfaz gráfica de usuario (GUI) en el que se muestran todos que constituyen la aplicación.

Atributos:

<<constant>> NUM_MAX_TRENES_EN_TRAMO = 5 => Número de trenes que se crean para que circulen por el tramo.

Asociaciones:

tramoAzul:TramoFerroviario => Contiene la instancia del tramo Azul definido en la red ferrroviaria.

tramoRojo:TramoFerroviario => Contiene la instancia del tramo Rojo definido en la red ferrroviaria.

elTunel:Tunel => Contiene la instancia del túnel de la red ferroviaria.

Constructor:

RedFerroviaria() => Constructor raíz de la red ferroviaria. Invoca recursivamente todos los demás constructores.

Métodos:

numMaxTrenesEnTramo() => Retorna el número máximo de trenes que pueden circular en el tramo. Está impuesto por la GUI y refleja la dimensión finita de los aparcamientos en él.

getTunel():Tunel => Retorna el acceso al objeto túnel de la red ferroviaria.

getTramoFerroviario(tramo:Tramos): TramoFerroviario => Retorna el acceso al objeto tramo ferroviario que es designado con el parámetro.

situaTren(tramo:Tramo, posicion:int) => Visualiza en la maqueta de la red ferroviaria un tren en la posición que se pasa como parámetro.

borraTren(tramo:Tramo, posicion:int) => Elimina de la maqueta de la red ferroviaria el tren situado en el tramo y posición que se pasan como parámetros.

situaTunel(numAzul:int,numRojo:int) => Visualiza en el estacionamiento del tunel un número de trenes igual a los que se pasan como parámetros.

situaCentroRegulacion(tramo:Tramo,numTrenes:int) => Visualiza en el estacionamiento del centro de regulación del tramo que se indica, el número de trenes que se pasa como parámetro.

Class **TramoFerroviario**: Representa un circuito que es seguido por un conjunto de trenes. Contiene un centro de regulación que controla su tráfico y comparte un túnel con otros tramos.

Atributos:

posicionLibre[] => Estado de cada posición de la red. Está libre si en ellas no está posicionado ningún tren. Los trenes ocupan una posición cuando visualizan su posición con situaTren() y la dejan libre cuando la borran con

borraTren().

tramo: Tramos => Identificador del tramo ferroviario que representa.

Asociaciones:

laRed: RedFerroviaria => Da acceso a la red ferroviaria y a sus servicios.

elCentroRegulación:CentroRegulacion => Contiene la instancia del centro de regulación que regula el tráfico de trenes en el tramo ferroviario.

Constructor:

TramoFerroviario(tramo:Tramos, laRed:RedFerroviaria) => Construye las instancias de los Tramos ferroviario. Se le pasan como parámetros su identificador de tramo y el acceso a la red ferroviaria.

Métodos:

esPosicionLibre(posición:int): boolean => Retorna True si la posición del tramo que se pasan comos parámetros no está ocupada por un tren. Se utiliza para evitar que los trenes adelanten. Las posiciones con aparcamiento (Centros de regulación y accesos al túnel) están siempre libres, ya que en ellos se pueden realizar adelantamientos.

enTunel(posicion:int): boolean => Retorna True si la posición que se pasa corresponde al interior del tunel.

enCentroRegulacion(posicion:int): Boolean => Retorna True si la posición del tren corresponde al centro de regulación.

posicionCentroRegulacion(): int => Retorna la posición del centro de regulación del tramo ferroviario que se pasa como parámetro.

longitudTramo(): int => Retorna la longitud del tramo ferroviario que se pasa como parámetro.

getCentroRegulacion():CentroRegulacion => Da acceso al centro de regulación del tramo ferroviario.

Tipos enumerados:

Enumerado **Tramos:** Tipo enumerado que da nombre a los dos tramos que contiene la red ferroviaria.

- TRAMO_ROJO: Designa al tramo ferroviario Rojo.
- TRAMO_AZUL: Designa al tramo ferroviario Azul.

Clase **Tren**: Describe cada uno de los trenes que se mueven en la red ferroviaria.

Atributos:

tramo:Tramos => Indica el tramo en el que circula el tren.

estado:EstadoTren => Representa el estado en el que se encuentra el tren.

posición: int => Posición que ocupa el tren dentro del tramo.

velocidad:int=> Velocidad a la que avanza el tren en pasos por segundos.

terminado:boolean =False => Vale True cuando se ha invocado termina() sobre el tramo ferroviario.

Asociaciones:

laRed: RedFerroviaria => Acceso a la red ferroviaria y a sus servicios.

Constructor:

Tren(tramo:Tramos, laRed:RedFerroviaria, int velocidad) => Constructor de la clase Tren. Hay que pasar el identificador del tramo en que circula, el acceso a la red ferroviaria y a los servicios que ofrece y la velocidad del tren.

Métodos:

actualiza() => Es invocado periódicamente por el programa principal para que cada objeto activo (tren) actualice su estado y situación. (Sólo se utiliza en la versión secuencial).

<<private>> **avanza()** => El tren avanza una posición en la vía si está libre. Visualiza la posición resultante en la red ferroviaria.

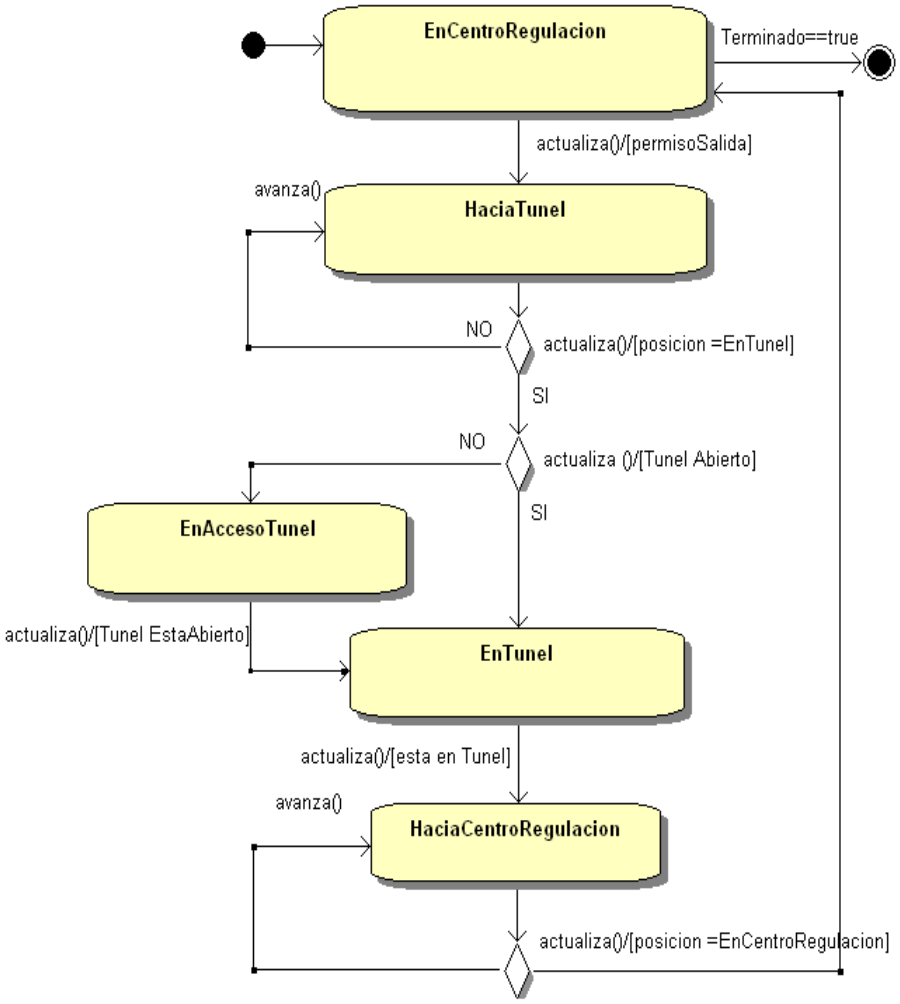
termina()=> El tren finaliza su recorrido.

Tipos enumerados:

Enumerado **EstadoTren** => Enumerado que define los estados en que se puede encontrar un tren:

EN_CENTRO_REGULACION
HACIA_TUNEL
EN_ACCESO_TUNEL
EN_TUNEL
HACIA_CENTRO_REGULACION

Comportamiento:



Class **CentroRegulacion**: Elemento que regula el tráfico por un tramo ferroviario. Controla el tiempo de salida de los trenes, y trata que el tráfico en el tramo sea regular.

Atributos:

numTrenes:int => Número de trenes que se encuentran estacionados en el centro de regulación.

proximaPartida:int => Tiempo en el que deberá partir el próximo tren. Se obtiene de acuerdo con la política de regulación que se utilice.

Asociaciones: **laRed: RedFerroviaria** => Da acceso a la red ferroviaria y a sus servicios.

Constructor:

CentroRegulacion(laRed:RedFerroviaria, tramo: Tramo) => Constructor de las instancias de centro de regulación. Recibe como parámetros el acceso a la red ferroviaria y a los servicios que ofrece, y el identificador del tramo en que se encuentra situado.

Métodos:

entro() => Un tren informa de que entra en el centro de regulación. Proporciona el tiempo que ha tardado su viaje.

salgo() => Lo invoca el tren que sale del centro de regulación.

permisoSalida():Boolean => Retorna True si el tren que lo invoca tiene autorización para salir del centro de regulación.

Class **Tunel**: Describe el túnel que es compartido por los trenes de los dos tramos ferroviarios. Es un tramo que solo puede ser utilizado concurrentemente por trenes que circulan en el mismo sentido.

Atributos:

numCirculando:int => Número de trenes que están circulando por el túnel.

autorizadoRojos: Boolean => Variable de estado de circulación por el túnel. Vale True si está autorizada la circulación de los trenes del tramo rojo.

rojosEstacionados:int => Número de trenes rojos estacionados en el correspondiente acceso al túnel.

azulesEstacionados:int => Número de trenes azules estacionados en el correspondiente acceso al túnel.

Asociaciones:

laRed: RedFerroviaria => Da acceso a la red ferroviaria y a sus servicios. A Través de él, el túnel visualiza los cambios en el estado de los dos Estacionamientos del túnel.

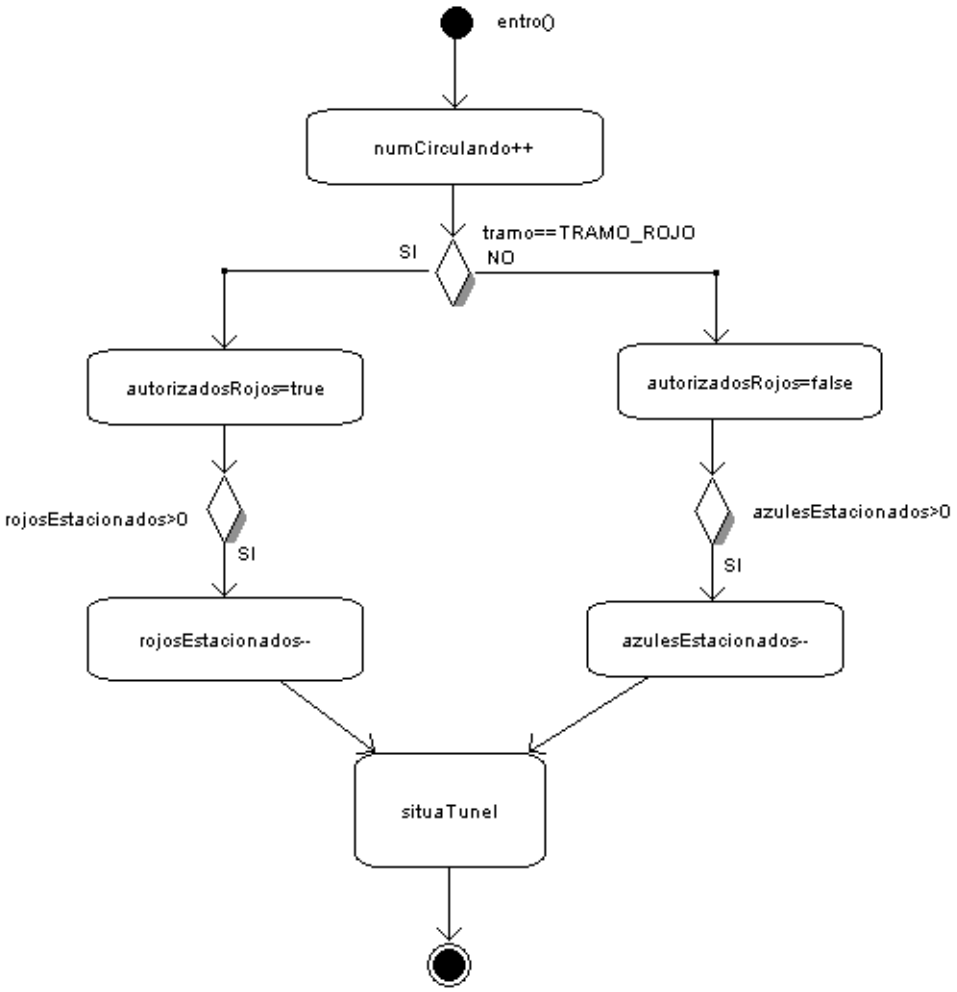
Constructor:

Tunel(laRed:RedFerroviaria) => Constructor de la clase Tunel. Recibe como parámetro el acceso a la red ferroviaria y a los servicios que ofrece.

Métodos:

salgo() => Lo invoca un tren cuando sale del túnel. Se Utiliza para llevar cuenta del número de trenes que está circulando por el tunel.

entro(tramo:Tramos) => Lo invoca un tren cuando entra a circular por el túnel.



estaciono(tramo:Tramos) => Lo invoca un tren cuando se estaciona en el acceso al túnel.

autorizadoEntrar(tramo:Tramos):Boolean => Retorna True si el tren que lo invoca está autorizado a entrar en el túnel en ese instante.

Ejemplo Implementación secuencial; RedFerroviariaSeq

Esta implementación secuencial es el punto de partida de la práctica, y se suministra como ejemplo de implementación en ejecución.

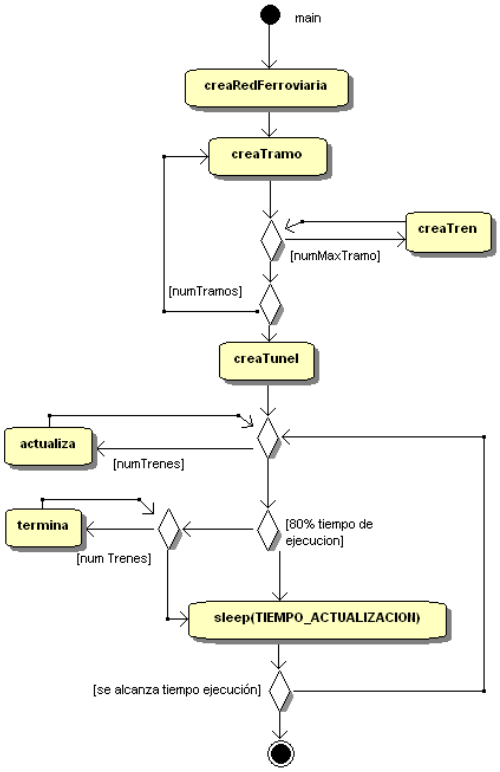
Para implementa la concurrencia propia de la aplicación utilizando un programa secuencialmente, se utiliza una estrategia de escrutinio periódico (polling) que consiste en que el thread principal (el único, es un programa secuencial) cada cierto tiempo (TIEMPO_ACTUALIZACION_MS actualmente establecido a 100 ms) invoca en cada tren la operación *actualiza()* para que decida lo que tiene que hacer en su situación. El programa principal controla el tiempo en que debe finalizar la actividad de los trenes.

Clase **PruebaSecuencial**: Programa principal que corresponde la ejecución de la implementación secuencial.

Métodos:

<<static>> **main(args:String)** => Procedimiento principal que lanza la aplicación. Realiza las siguientes tareas:

- Instancia una redFerroviaria (y recursivamente los elementos agregados en ella).
- Instancia los trenes que circulan por cada red ferroviaria.
- Mientras no se cumpla el TIEMPO_DE_EJECUCION invoca el método *actualiza()* de cada tren.
- Si el tiempo actual es mayor o igual que el 80% del TIEMPO_DE_EJECUCION, se invoca el método *termina()* sobre cada tren, que cambia el valor de *terminado* de tren a true.



Práctica Implementación concurrente RedFerroviariaConcurrente

Especificaciones:

Esta es una versión en la que **se utilizan los Threads Java** para implementar los **trenes y los lock** intrínsecos de la clase Object de Java para implementar las clases **Tunel y CentroRegulacion**.

Todo debe funcionar igual que en TunelSecuencial salvo que:

- El main() instancia la Red y los trenes.
- Los trenes terminan su recorrido ordenadamente y aparcando en el CentroRegulacion cuando sobre ellos se ejecute *interrupt()*.
- La función actualiza() de la clase Tren, deja de utilizarse. El movimiento de los trenes es autónomo, conducido por su thread interno.
- La función *entro()* de la clase Tunel debe suspender el thread del tren que lo invoca, hasta que el túnel autorice su acceso, y el thread del tren estacionado se activa de nuevo.
- La función *entro()* de la clase CentroRegulacion debe suspender el thread del tren que lo invoca, hasta que el centro de regulación autorice su salida, y el thread del tren estacionado se activa de nuevo.