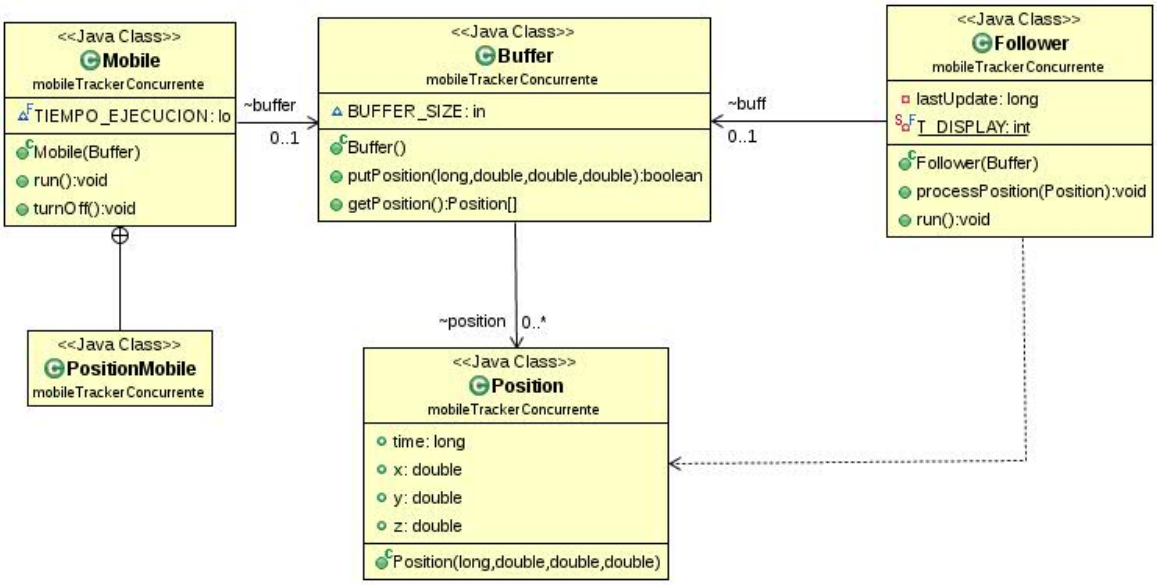
Programación Distribuida
Grado de Ingeniería Informática

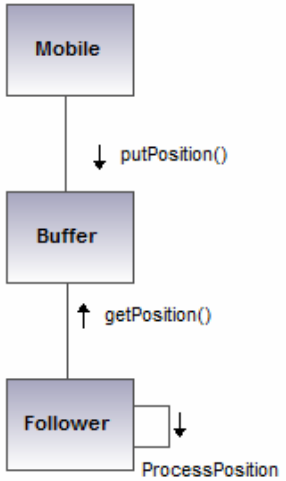
Se dispone de un sistema de **seguimiento de la posición de un móvil**, compuesto por tres clases: El móvil (Mobile) que cada 250 ms lee por el GPS su posición y la almacena en un **Buffer** con capacidad de 20 posiciones. El **Controlador** que lee del buffer esporádicamente los datos de posición del móvil, y en función de los datos que lee, muestra la trayectoria en la consola en base a una posición cada 2 s.

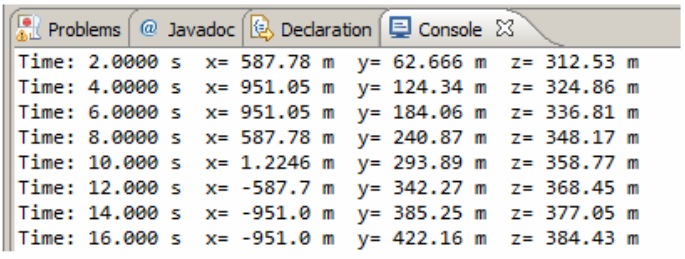
Cuando se ejecuta la aplicación, el móvil va introduciendo las posiciones en el buffer. Si el buffer se llena (porque el Follower en su actividad esporádica tarda demasiado tiempo en leer el buffer), el móvil recibe de retorno false, por lo que el móvil debe almacenarlo internamente sin parar de leer periódicamente sus posiciones, para trnaferirlo cuando haya hueco en el buffer.

La iniciativa de finalizar la aplicación la toma el objeto de la clase Mobile. Ésta decide finalizar y termina, dejando de transmitir datos al buffer. El objeto de la clase Follower, cuando accede a al objeto de la clase Buffer y no encuentra datos, también termina y con él la aplicación.



Cuando la aplicación se ejecuta, aparece esporádicamente en la ventana de consola las posiciones obtenidas por el móvil cada 2 segundos. Cuando transcurren 50 segundos la aplicación finaliza definitivamente (todo los threads terminan correctamente).

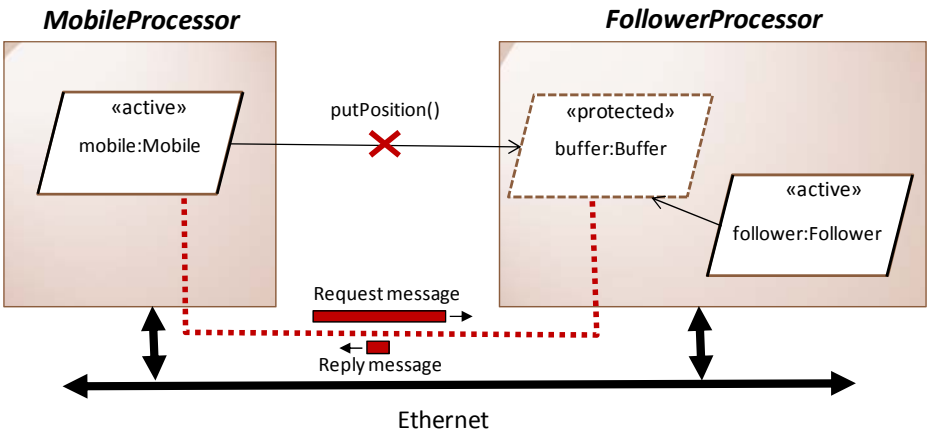




En el fichero MobileTracker_Sockets.zip se proporciona la versión original (concurrente), así como la implementación distribuida mediante sockets TCP y UDP. En lo sucesivo, este documento se centra en la distribución con TCP, pero puede utilizarse un razonamiento similar para la distribución con UDP.

Versión Distribuida

La aplicación se rediseña para ser ejecutada como dos particiones en dos procesadores: La primera *mobilePartition* contiene el objeto de la clase Mobile que actúa como cliente. La segunda *followerPartition* contiene los objetos de las clases Buffer y Follower, y actúa como servidor. Los dos nodos MobileProcessor y FollowerProcessor están conectados físicamente mediante Ethernet y se comunican a través de sockets.

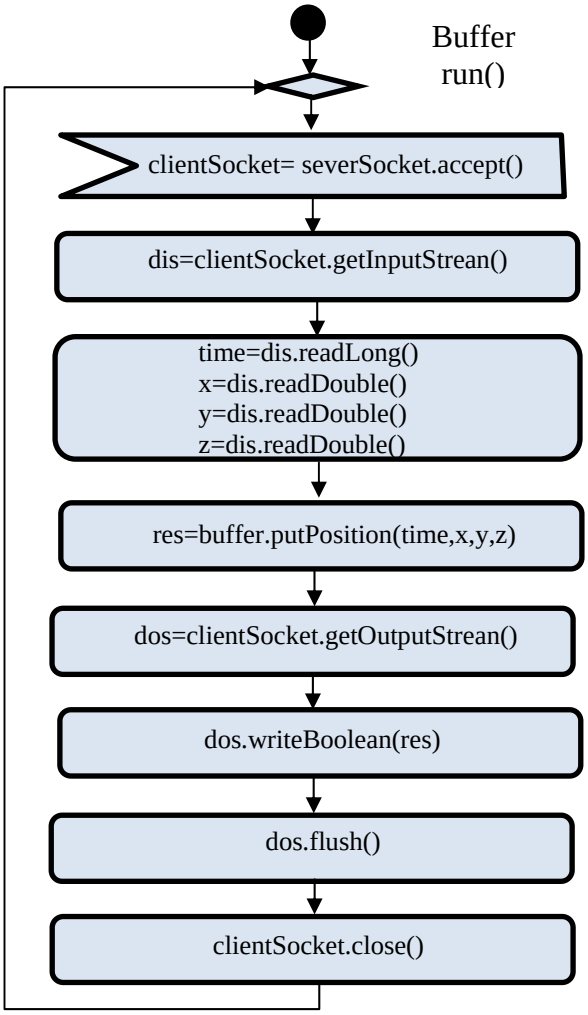


MobilePartition: Representa una aplicación que puede ser ejecutada independientemente en el procesador que se desee. Al actuar como cliente, debe saber el IP en el que se instala la partición servidor (en el código se asume que se utiliza el mismo procesador IP=[127,0,0,1]), y el puerto por el que atiende (en el código por el puerto 12000):

- Clase principal **MobileLauncher**: Su ejecución en el computador MobileProcessor lanza la ejecución de MobilePartition.
- Clase de negocio **Mobile**: Implementa la funcionalidad del móvil. Utiliza un socket para comunicarse con la partición remota, que es quien por delegación hace la invocación del método putPosition() sobre el objeto de negocio Buffer, y del que recibe el resultado booleano que retorna.

FollowerPartition: Es una aplicación que puede ser ejecutada independientemente, y que contiene:

- Clase principal **FollowerLauncher**: Su ejecución en el computador FollowerProcessor lanza la followerPartition.
- Clase de negocio **Follower**: Implementa la funcionalidad del objeto que monitoriza la posición del móvil. Su código no ha sido modificado respecto del proyecto original.
- Clase de negocio **Buffer**: Implementa la funcionalidad del objeto que sirve de enlace entre el móvil y el monitor. Es un objeto activo que permanece atento a los socket que tratan de conectarse con el puerto 12000. Cuando la partición Mobile trata de conectarse con él, lo acepta, y lee el mensaje que se recibe que tiene que corresponder a una invocación remota del método putPosition() del objeto Buffer. Del mensaje extrae el valor de los parámetros *time:*long, *x:*double, *y:*double, *z:*double; con ellos se invoca el método, y con el resultado booleano que retorna el método se construye un mensaje que se envía por el mismo socket en la dirección opuesta. Al ser una clase activa, el método run () define la actividad del thread de acuerdo al siguiente diagrama de actividad:



Se repite por cada invocación

Se espera a que el proxy cliente crea el socket

Obtiene el stream de entrada del socket

Lee del stream los cuatro valores de la Invocación.

Realiza la invocación sobre el objeto de negocio buffer u obtiene el resultado.

Se obtiene el stream de salida del socket

Se escribe el valor booleano de retirno

Se ordena su envío inmediato

Se cierra el socket.