

Sistemas Distribuidos

**PR 2 – Report for Phases 2-4**



Raúl Pastor Clemente

[*rpastorc@uoc.edu*](mailto:rpastorc@uoc.edu)

Group ID 309571699

Contenido

[Introducción 2](#_Toc40472801)

[Implementación 4](#_Toc40472802)

[Fase 2 8](#_Toc40472803)

[Pruebas locales (configuración de ejecución en Eclipse) 8](#_Toc40472804)

[Pruebas remotas (ejecución en dslab) 8](#_Toc40472805)

[Fase 3 9](#_Toc40472806)

[Pruebas locales (configuración de ejecución en Eclipse) 9](#_Toc40472807)

[Pruebas remotas (ejecución en dslab) 9](#_Toc40472808)

[Fase 4.1 10](#_Toc40472809)

[Pruebas locales (configuración de ejecución en Eclipse) 10](#_Toc40472810)

[Pruebas remotas (ejecución en dslab) 10](#_Toc40472811)

[Dificultades 11](#_Toc40472812)

[Anexo 1: Configuración del entorno de pruebas 12](#_Toc40472813)

[Anexo 2: Métodos TSAESession\* sessionTSAE() y run() 14](#_Toc40472814)

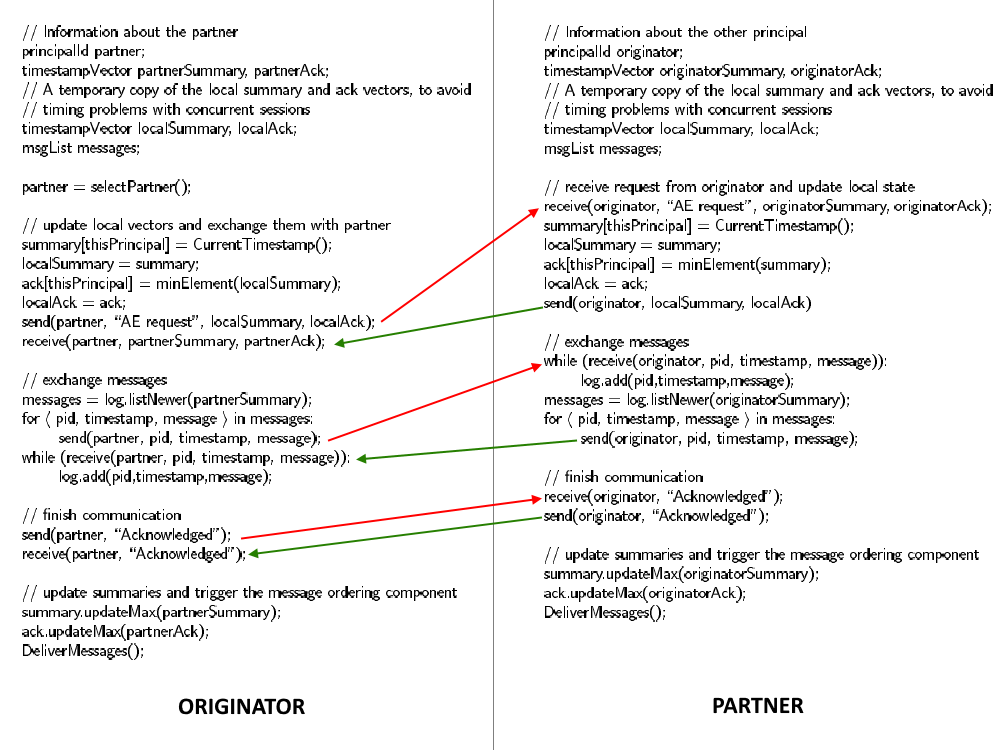
# Introducción

Esta práctica trata de completar la implementación de una simulación del protocolo TSAE en Java. Las clases que hay que implementar en esta práctica son:

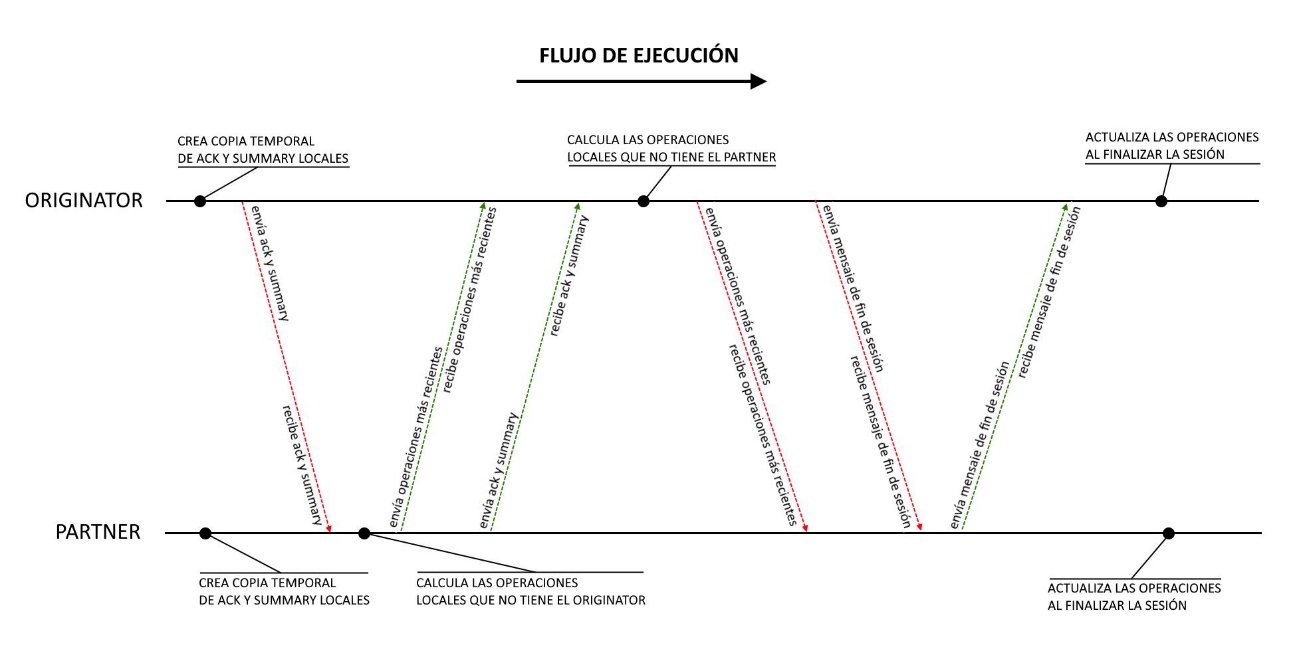
* **Log.java**: esta clase se encarga de almacenar los logs de las operaciones. Contiene todos los métodos necesarios para su manipulación.
* **TimestampVector.java**: esta clase se encarga de almacenar el resumen (summary) del estado de los nodos. Un TimestampVector está compuesto de
* **TimestampMatrix.java**: esta clase almacena la matrix de acknowledge. Se trata de una matriz compuesta por vectores de tiempo pertenecientes a los distintos nodos.
* **ServerData.java**: controla el acceso a los datos. Los métodos que más nos interesan son addRecipe() y removeRecipe().
* **TSAESession\*Side.java**: se trata de las dos clases más importantes, ya que ambas se se encargan de establecer una sesión TSAE entre dos nodos. Una de las clases es para el originador de la sesión, y la otra para el partner.

|  |
| --- |
|  |

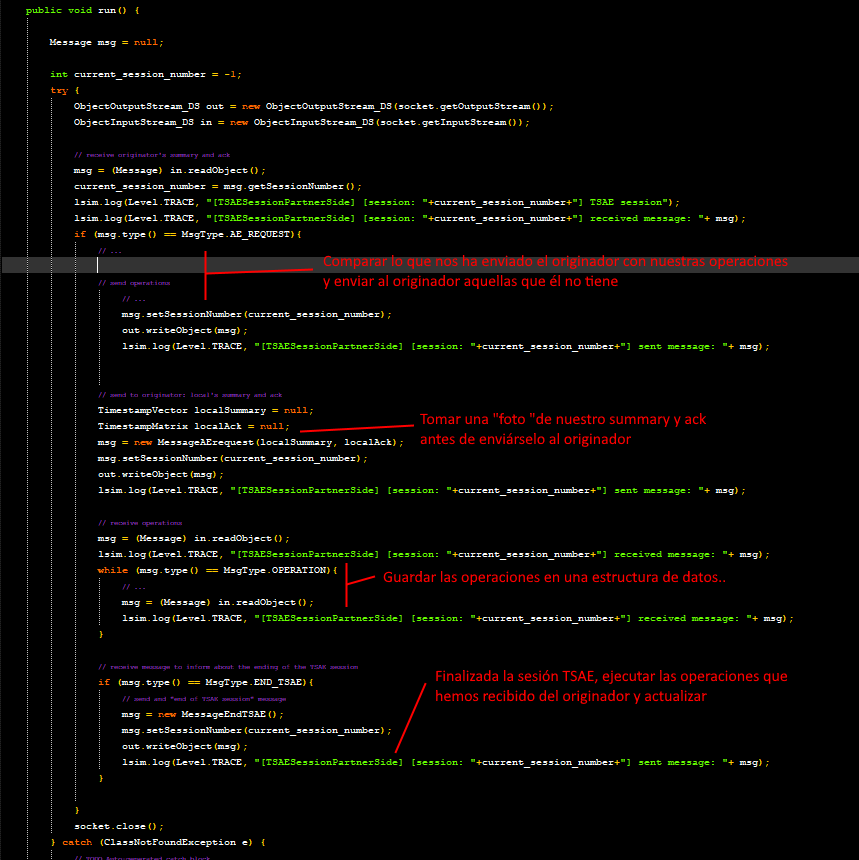
Para entender bien la dinámica que hay entre estas dos clases es necesario recurrir al *paper* de Golding para analizar cómo y cuándo se comunican entre ellas. En las páginas 57 y 58 de dicho paper se encuentra una representación en pseudocódigo de la estructura de una sesión:



Si comparamos este pseudocódigo con las clases incompletas que nos han suministrado en el proyecto, resulta más sencillo ver cuáles son las partes faltan y qué deberían hacer. La estructura que siguen en esta implementación es casi idéntica a la de Golding:



Un ejemplo de este análisis aplicado al método *run*() de la clase *TSAESessionPartnerSide*:



De esta forma, sabiendo cómo se comunican los nodos y qué tipo de estructuras se pasan entre ellos, podemos **pasar a la** **fase de implementación**.

# Implementación

Lo primero, antes de comenzar, fue ver qué métodos se usaban en cada clase. Para ello observé las ejecuciones desde el entorno Eclipse:

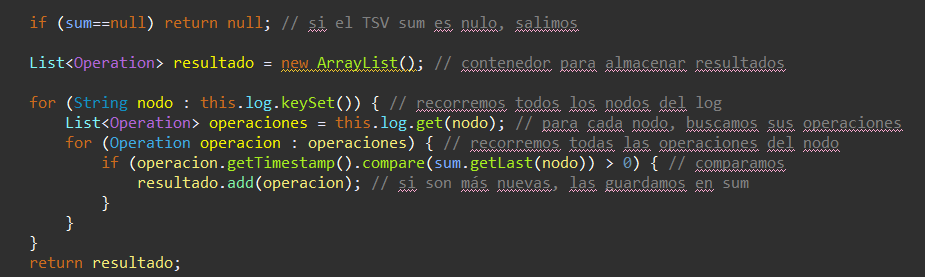
|  |  |  |  |
| --- | --- | --- | --- |
| Clase | Fase 2 | Fase 3 | Fase 4 |
| Log.java | *listNewer*() | *purgeLog*() |  |
| TimestampVector.java | *clone*()  *getLast*()  *mergeMin*() *updateMax*() |  |  |
| ServerData.java | *recipeManager*()\* |  | *removeRecipe*() |
| TSAESessionOriginatorSide.java | *sessionTSAE*() |  |  |
| TSAESessionPartnerSide.java | *run*() |  |  |
| TimestampMatrix.java | No es necesario implementar esta clase en la fase 2, pero *equals*() debe modificarse para que siempre devuelva *true* | *clone*()  *equals*()  getTimestampVector()  *minTimestampVector*()  *updateMax*()  *update*() |  |

*\* método propio*

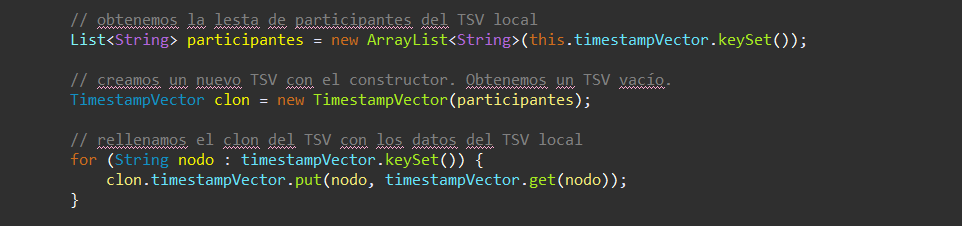
Sin duda, la **fase 2** es la más complicada de todas. No solo hay que entender cómo funciona una sesión TSAE, sino que además hay que implementar correctamente los métodos de *Log* y *TimestampVector* necesarios para la manipulación de las estructuras de datos necesarias. Además, estaba la “trampa” de que *TimestampMatrix.equals* devolvía *false* por defecto, siendo necesario cambiar este valor a *true* para que funcionara. Por último, había que implementar “algo” en *ServerData* que permitiera añadir las recetas obtenidas durante la sesión TSAE. Ese “algo” en mi caso es un método al que he llamado *recipeManager*(*MessageOperation* mso).

Las funciones que se han implementado son:

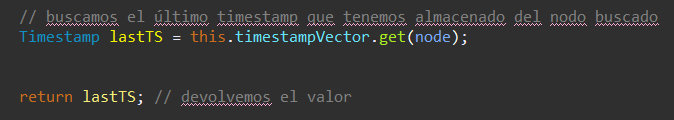
* **Log.listNewer**: esta función compara los logs de dos nodos y devuelve las operaciones que el nodo local no tiene, pero el remoto sí. Se usa en las dos clases TSAESession\*.



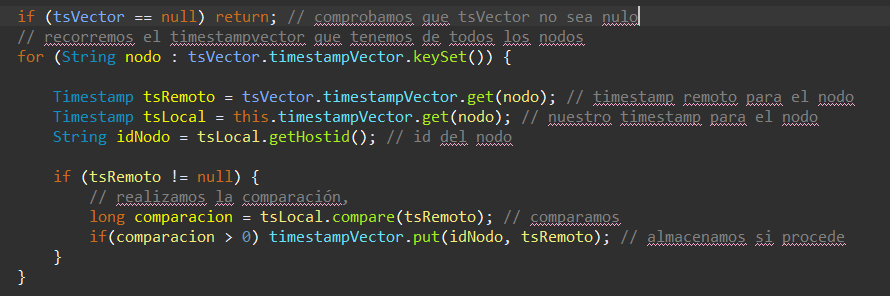
* **TimestampVector.clone**: esta función devuelve una copia del TimestampVector. Se emplea durante la sesión TSAE para tomar una “foto” estática del estado del nodo. Se usa el constructor de la clase para crear un nuevo TSV y después se rellena con datos.



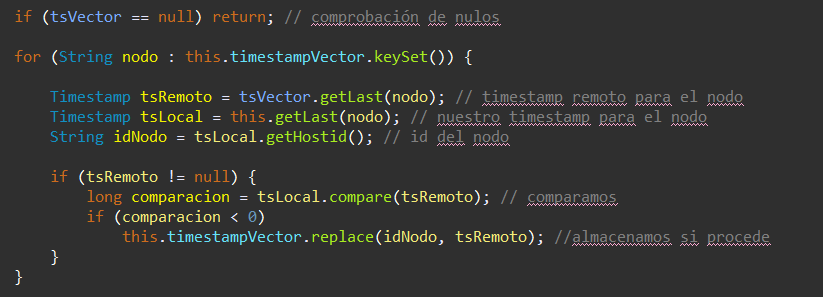
* **TimestampVector.getLast**: este método devuelve el último *Timestamp* del nodo que se le pasa como parámetro.



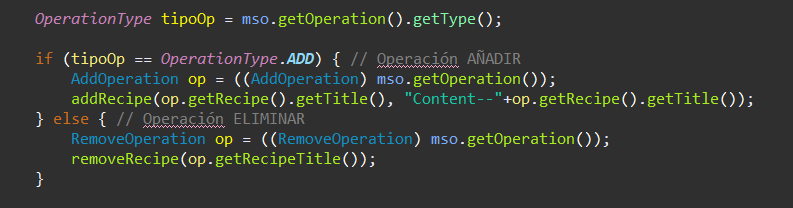
* **TimestampVector.mergeMin**: fusiona dos *TSV* con los elementos mínimos de ambos.



* **TimestampVector.updateMax**: fusiona dos *TSV* con los elementos máximos de ambos.



* **ServerData.recipeManager**: se encarga de gestionar las operaciones de adición y eliminación de recetas. Al recibir un MessageOperation se determina el tipo de operación que contiene y se trata de una forma u otra.

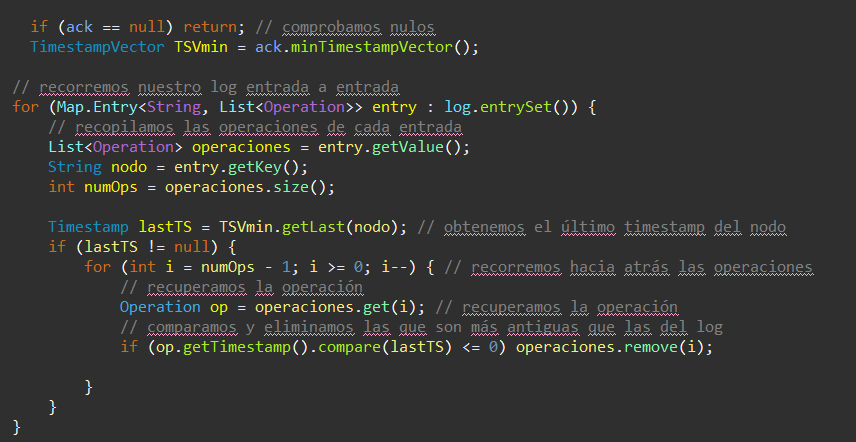


* **TSAESessionOriginatorSide.sessionTSAE**: esta función es necesaria para establecer una sesión TSAE, y es la que usa el originador. El código está en el **anexo 3**.
* **TSAESessionPartnetSide.run**: igual que la anterior, pero ésta es la parte usada por el *partner* en la sesión TSAE. El código está en el **anexo 3**.
* **TimestampMatrix.equals**: en esta función solo se ha cambiado el retorno a “*true*” para que la práctica pueda funcionar.

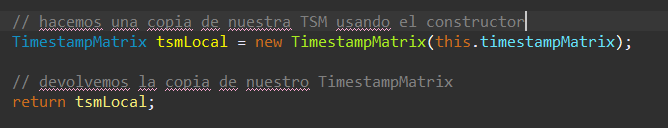
La **fase 3** no fue tan complicada como la anterior. Era necesario implementar la purga de *logs* y la clase *TimestampMatrix*, que almacena los *acks*. Afortunadamente, los métodos de *TimestampMatrix* no distan mucho de los empleados en *TimestampVector*, y además se ha podido reutilizar código entre clases e incluso métodos de la propia clase.

Las funciones que se han implementado son:

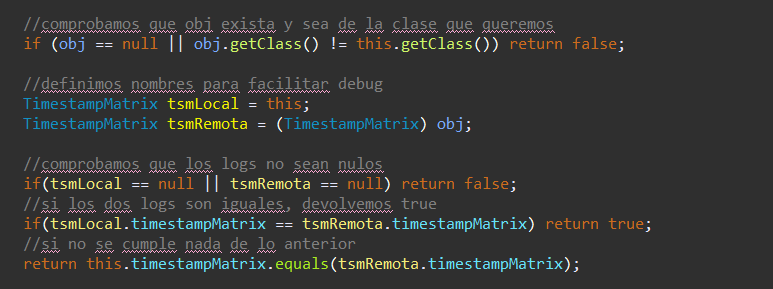
* **Log.purgeLog**: elimina del *log* las operaciones que tienen todos los nodos.



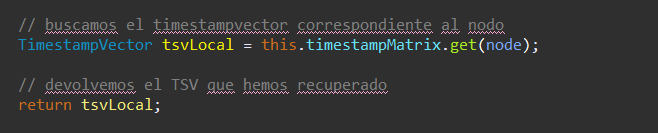
* **TimestampMatrix.clone**: este método devuelve una copia de la matriz.



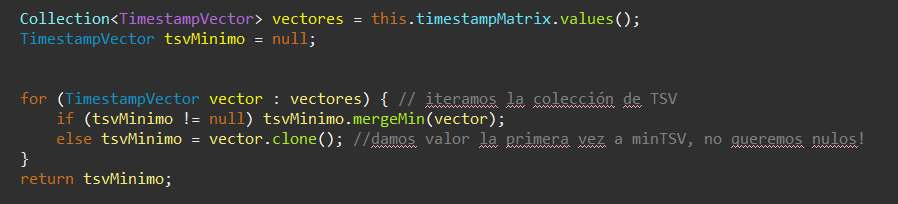
* **TimestampMatrix.equals**: comprueba si dos estructuras *TimestampMatrix* son iguales, y devuelve *true* si lo son o *false* si son diferentes.



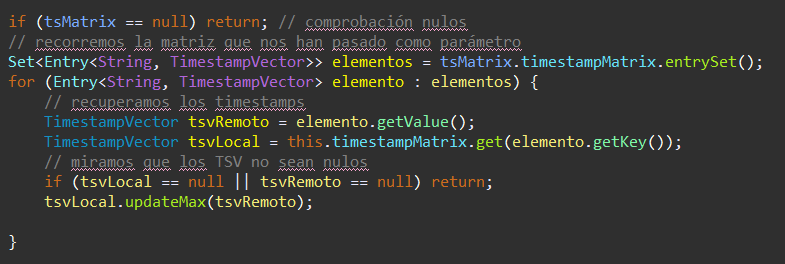
* **TimestampMatrix.getTimestampVector**: devuelve el *TimestampVector* de un nodo.



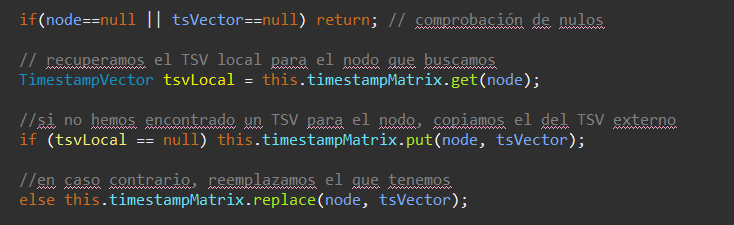
* **TimestampMatrix.minTimestampVector**: devuelve un *TimestampVector* con los *Timestamp* conocidos por todos los participantes.



* **TimestampMatrix.updateMax**: Toma dos *TimestampMatrix* y crea una nueva con los *Timestamps* más recientes.



* **TimestampMatrix.update**: sustituye el *TimestampVector* de un nodo por otro:



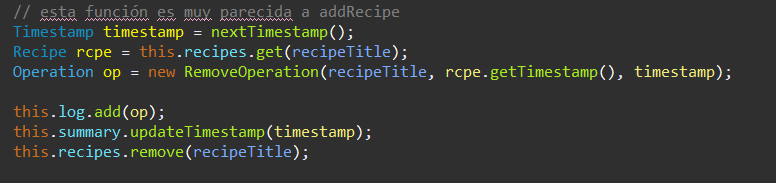
Sin embargo, la **fase 4** sí que fue problemática, puesto que nos encontramos ante una situación en la que los nodos, al desconectase y volverse a conectar al rato, contienen información desactualizada y pueden provocar el caos, puesto que:

* Se puede dar el caso de que un nodo trate de eliminar una receta que no tiene
* También puede darse el caso de que agregue una receta que ya había eliminado previamente.

Así, nos encontramos con que la solución que nos funcionaba perfectamente en las fases 2 y 3 ahora da problemas, una vez hemos implementado la eliminación de recetas en *ServerData*.

Las funciones que se han implementado son:

* **ServerData.removeRecipe**: este método elimina una receta, generando un nuevo Timestamp para la operación y actualizando *log* y el *summary*. La utilizan las clases *Server* y *ActivitySimulation* durante la actividad de simulación.



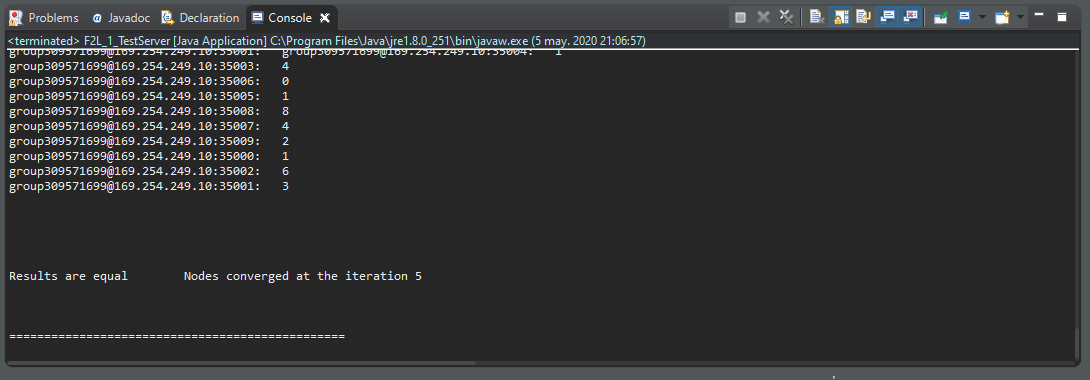
# Fase 2

En esta fase, como ya he explicado, se deben hacer modificaciones en 5 de las 6 clases de Java de la práctica. Una vez hecho esto, he realizado pruebas en local directamente desde Eclipse y en dslab. La configuración de eclipse se refiere a los grupos de ejecución usados (ver anexo 1):

## Pruebas locales (configuración de ejecución en Eclipse)

**Configuración Eclipse**

* **Run Configurations (Java Application)**
  + **Nombre**: F2\_1\_TestServer
    - **Main Class**: recipes\_service.test.TestServer
    - **Arguments**: 20004 10 --logResults -path ../results --nopurge --noremove
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F2\_2\_SendArgsToServer
    - **Main Class**: recipes\_service.test.SendArgsToTestServer
    - **Arguments**: 20004 10 --logResults -path ../results --nopurge --noremove
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F2\_3\_Recipes
    - **Main Class**: recipes\_service.Server
    - **Arguments**: 20004 10 --logResults -path ../results --nopurge --noremove
* **Launch Group**
  + **Nombre**: FASE 2
    - **application**: F2\_1\_TestServer, mode: run, action: delay 2 seconds
    - **application**: F2\_2\_SendArgsToServer, mode: run, action: delay 2 seconds
    - **application**: F2\_3\_Recipes, mode: run, action: none (x10 instancias)

**Resultado:**

## Pruebas remotas (ejecución en dslab)

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Project | Execution date | Result |
| 1442 | PR2\_P2\_V04 | 2020-05-15 08:22 | ✔️ Nodes converged at the iteration 1 |
| 1442 | PR2\_P2\_V04 | 2020-05-15 08:29 | ✔️ Nodes converged at the iteration 1 |
| 1442 | PR2\_P2\_V04 | 2020-05-15 08:36 | ✔️ Nodes converged at the iteration 1 |
| 1442 | PR2\_P2\_V04 | 2020-05-15 08:57 | ✔️ Nodes converged at the iteration 1 |
| 1442 | PR2\_P2\_V04 | 2020-05-15 09:04 | ✔️ Nodes converged at the iteration 1 |

**Captura de dslab:**



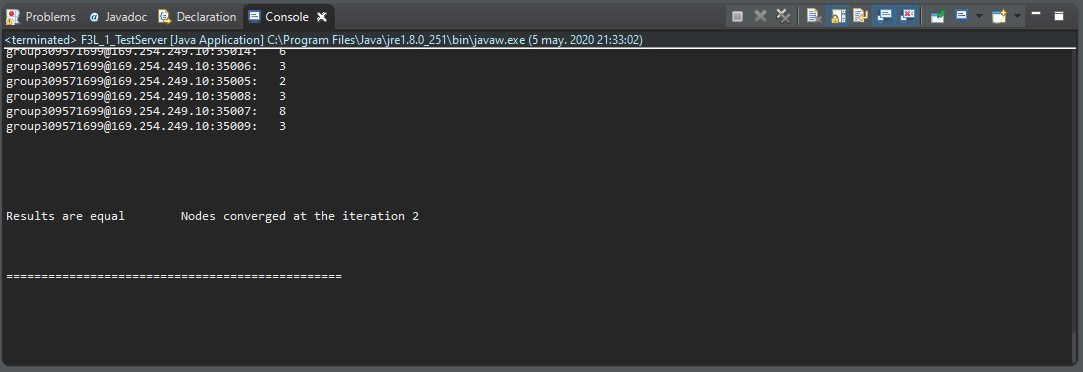
# Fase 3

Esta fase se añade la purga de logs, y la matriz de *acknowledges* TimestampMatrix.java.

## Pruebas locales (configuración de ejecución en Eclipse)

**Configuración Eclipse**

* **Run Configurations (Java Application)**
  + **Nombre**: F3\_1\_TestServer
    - **Main Class**: recipes\_service.test.TestServer
    - **Arguments**: 20004 15 --logResults -path ../results --noremove
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F3\_2\_SendArgsToServer
    - **Main Class**: recipes\_service.test.SendArgsToTestServer
    - **Arguments**: 20004 15 --logResults -path ../results --noremove
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F3\_3\_Recipes
    - **Main Class**: recipes\_service.Server
    - **Arguments**: 20004 15 --logResults -path ../results --noremove
* **Launch Group**
  + **Nombre**: FASE 3
    - **application**: F3\_1\_TestServer, mode: run, action: delay 2 seconds
    - **application**: F3\_2\_SendArgsToServer, mode: run, action: delay 2 seconds
    - **application**: F3\_3\_Recipes, mode: run, action: none (x15 instancias)

**Resultado:**

## Pruebas remotas (ejecución en dslab)

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Project | Execution date | Result |
| 1444 | PR2\_P3\_v08 | 2020-05-15 09:13 | ✔️ Nodes converged at the iteration 1 |
| 1444 | PR2\_P3\_v08 | 2020-05-15 09:21 | ✔️ Nodes converged at the iteration 1 |
| 1444 | PR2\_P3\_v08 | 2020-05-15 09:33 | ✔️ Nodes converged at the iteration 3 |
| 1444 | PR2\_P3\_v08 | 2020-05-15 09:41 | ✔️ Nodes converged at the iteration 1 |
| 1444 | PR2\_P3\_v08 | 2020-05-15 09:56 | ✔️ Nodes converged at the last iteration |
| 1444 | PR2\_P3\_v08 | 2020-05-15 10:09 | ✔️ Nodes converged at the iteration 1 |

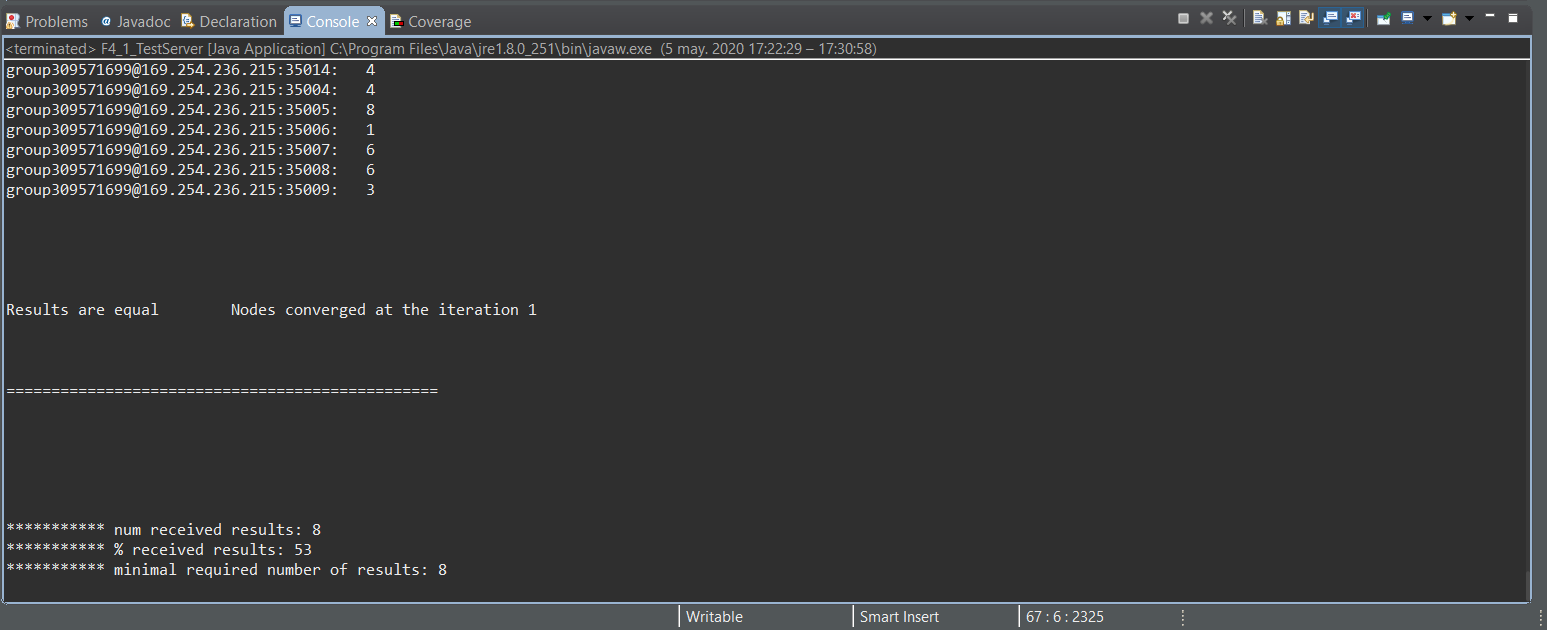
**Captura de dslab:**



# Fase 4.1

## Pruebas locales (configuración de ejecución en Eclipse)

* **Run Configurations (Java Application)**
  + **Nombre**: F4\_1\_TestServer
    - **Main Class**: recipes\_service.test.TestServer
    - **Arguments**: 20004 15 --logResults -path ../results
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F4\_2\_SendArgsToServer
    - **Main Class**: recipes\_service.test.SendArgsToTestServer
    - **Arguments**: 20004 15 --logResults -path ../results
    - **Working Directory**: ${workspace\_loc:SSDD\_v3/scripts}
  + **Nombre**: F4\_3\_Recipes
    - **Main Class**: recipes\_service.Server
    - **Arguments**: 20004 15 --logResults -path ../results
* **Launch Group**
  + **Nombre**: FASE 4
    - **application**: F4\_1\_TestServer, mode: run, action: delay 2 seconds
    - **application**: F4\_2\_SendArgsToServer, mode: run, action: delay 2 seconds
    - **application**: F4\_3\_Recipes, mode: run, action: none (x15 instancias)

**Resultado:**

## Pruebas remotas (ejecución en dslab)

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Project | Execution date | Result |
| 1057 | PR2\_P4\_v06 | 2020-05-11 18:33 | ❌ Results are NOT equal |
| 1057 | PR2\_P4\_v06 | 2020-05-11 18:40 | ❌ Results are NOT equal |
| 1057 | PR2\_P4\_v06 | 2020-05-11 17:54 | ❌ Results are NOT equal |
| 1057 | PR2\_P4\_v06 | 2020-05-11 17:54 | ✔️ Nodes converged at the last iteration |
| 1057 | PR2\_P4\_v06 | 2020-05-11 17:54 | ❌ Results are NOT equal |

**Captura de dslab:**



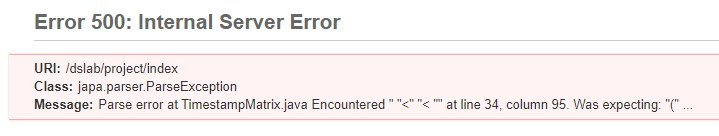
# Dificultades

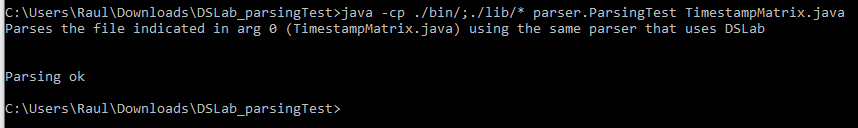
Hay que decir que, en conjunto, esta práctica ha sido bastante complicada.

La **primera dificultad** fue **configurar el entorno de desarrollo**. Para la fase 1 utilicé una máquina virtual con Ubuntu 14.04 y OpenJDK 7 en VirtualBox, pero se trataba de una solución lenta, engorrosa y pesada, ya que ocupada unos 10 GB.

Así pues, para la segunda práctica decidí intentarlo en Windows 10 con éxito. En el anexo 1 se detalla cómo configuré el entorno de desarrollo para que el proyecto funcionara.

La **segunda dificultad** fueron los **errores derivados de usar Java 8**. Esta versión presenta algunas diferencias con Java 7. Esto provocó que en local me funcionara pero que me diera errores al compilar el proyecto en DSLAB. Pude solucionarlo usando el *parser* *offline* descargable. Estaría bien actualizar el proyecto para que funcionara con versiones más recientes de Java.





Otra **dificultad** es que **la práctica está muy orientada a entornos Linux**, a pesar de que Java es un lenguaje multiplataforma. Las instrucciones del *parser* solo funcionan en Linux, y en la práctica solo se proporciona un *shell script* para las pruebas en local. Sería deseable la elaboración de un archivo *batch* para que los usuarios del S.O. de Microsoft pudieran ejecutar las pruebas.

En cuanto al **entorno de pruebas *dslab***, a veces no estaba accesible, en ocasiones los experimentos se quedaban atascados, y raras veces recibo el correo con los resultados de las pruebas.

Por supuesto, lo más difícil de todo fue **entender el funcionamiento del protocolo TSAE**. El *paper* de Golding explica bien las cosas, pero es bastante largo y denso. Por suerte, en la práctica solo se tocan las sesiones anti-entropía y las estructuras de datos necesarias, y además contaba con el *debugger* de Eclipse para inspeccionar el funcionamiento del protocolo.

# Anexo 1: Configuración del entorno de pruebas

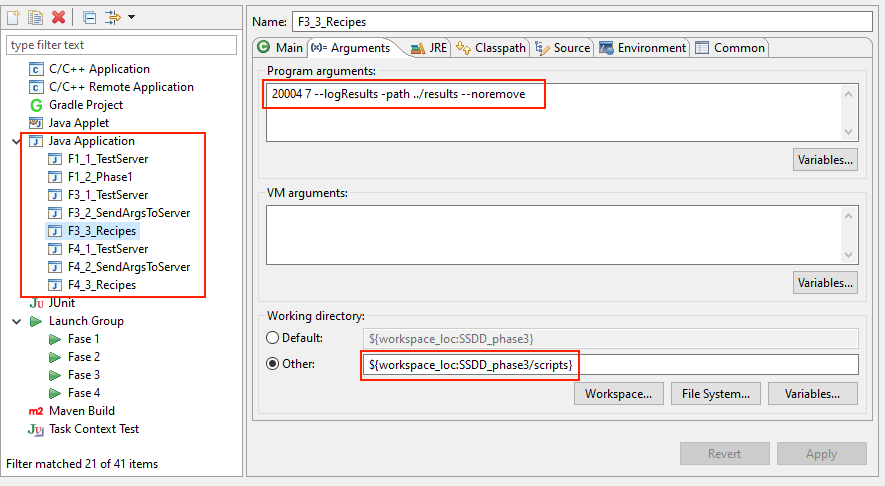
Utilicé el entorno de desarrollo Eclipse Neon 3 en Windows 10 1909.

Los componentes usados son:

* Java JDK 1.8.0\_251-b08 de 64-bits.
* Eclipse Neon 3 versión 4.6.3, edición para Java Developers (también funciona en versiones más recientes).
* Plugin de Eclipse “C/C++ Remote (over TCF/TE) Run/Debug Launcher”.

El principal problema de la práctica es que está pensada para realizarse bajo entornos Linux. Así, tuve que analizar el contenido del fichero “start.sh” e instalar un plugin de lanzamiento remoto que sirve para lanzar secuencialmente varias ejecuciones y que, a pesar de su nombre, no se limita a C y funciona con cualquier lenguaje.

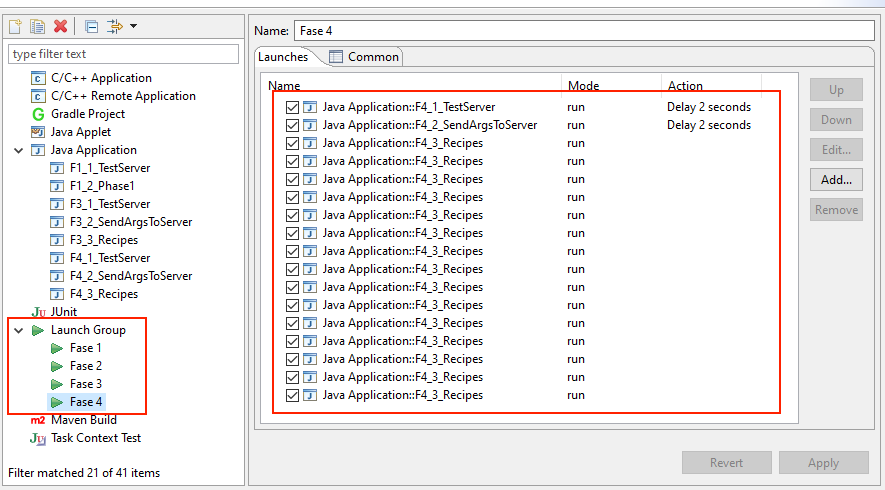
Para cada fase tuve que recrear configuraciones de ejecución para cada componente con los mismos argumentos que introduciría en el *script*:



Se han creado distintas configuraciones de ejecución porque los argumentos varían entre fases.

Fue necesario que todas las configuraciones de ejecución tomaran como base el directorio “/scripts”, pues de lo contrario la ejecución daba un fallo por no encontrar en directorio “../lsimLogs”.

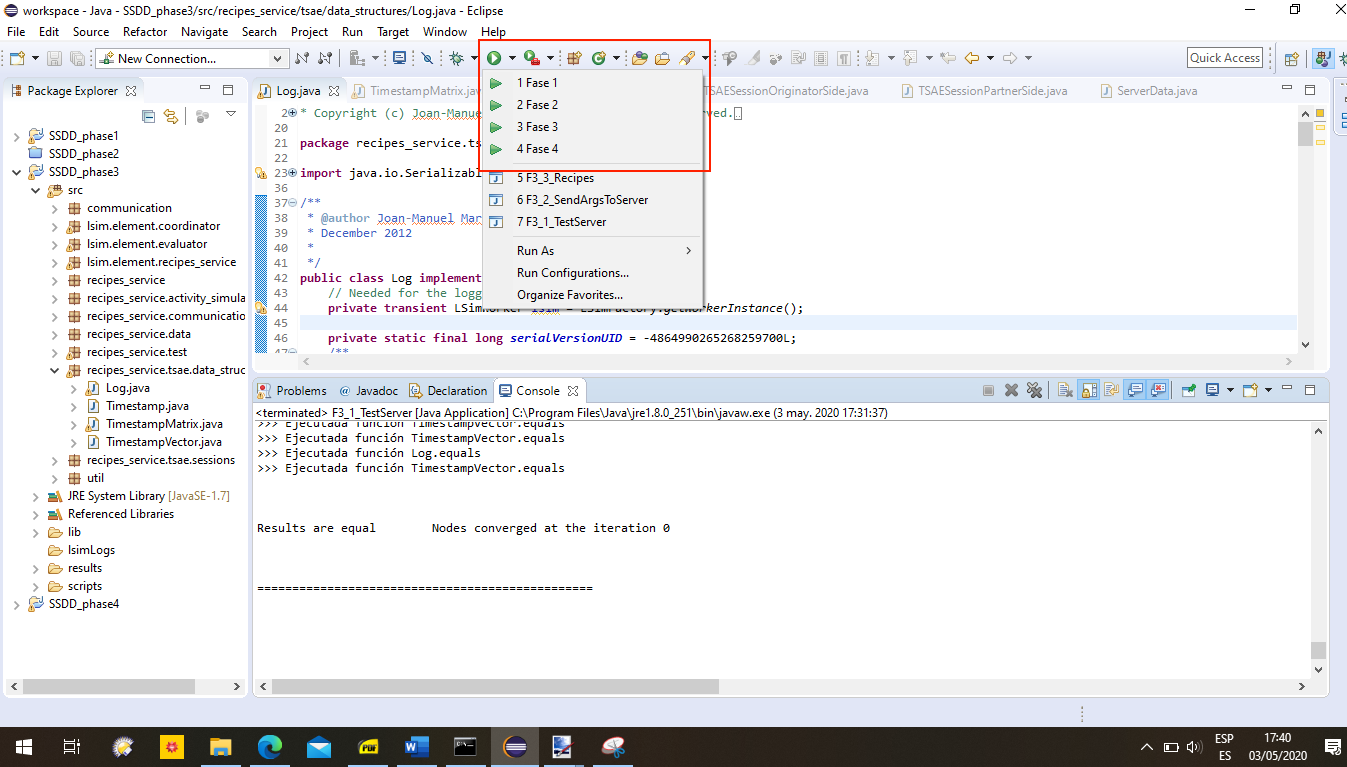
Una vez recreadas las ejecuciones de las distintas clases, lo siguiente fue crear los grupos de lanzamiento de todas las partes del experimente utilizando para ello el plugin de lanzamiento remoto, tal y como se lanzan desde el *script shell*:



En el recuadro rojo aparecen los cuatro grupos de ejecución que se corresponden con las cuatro fases.

Una de las cosas que permite el plugin de lanzamiento es ejecutar una acción tras realizar un lanzamiento. Gracias a esto, puedo introducir un retraso entre el lanzamiento del servidor local y el resto de los componentes, como pasa en el script.

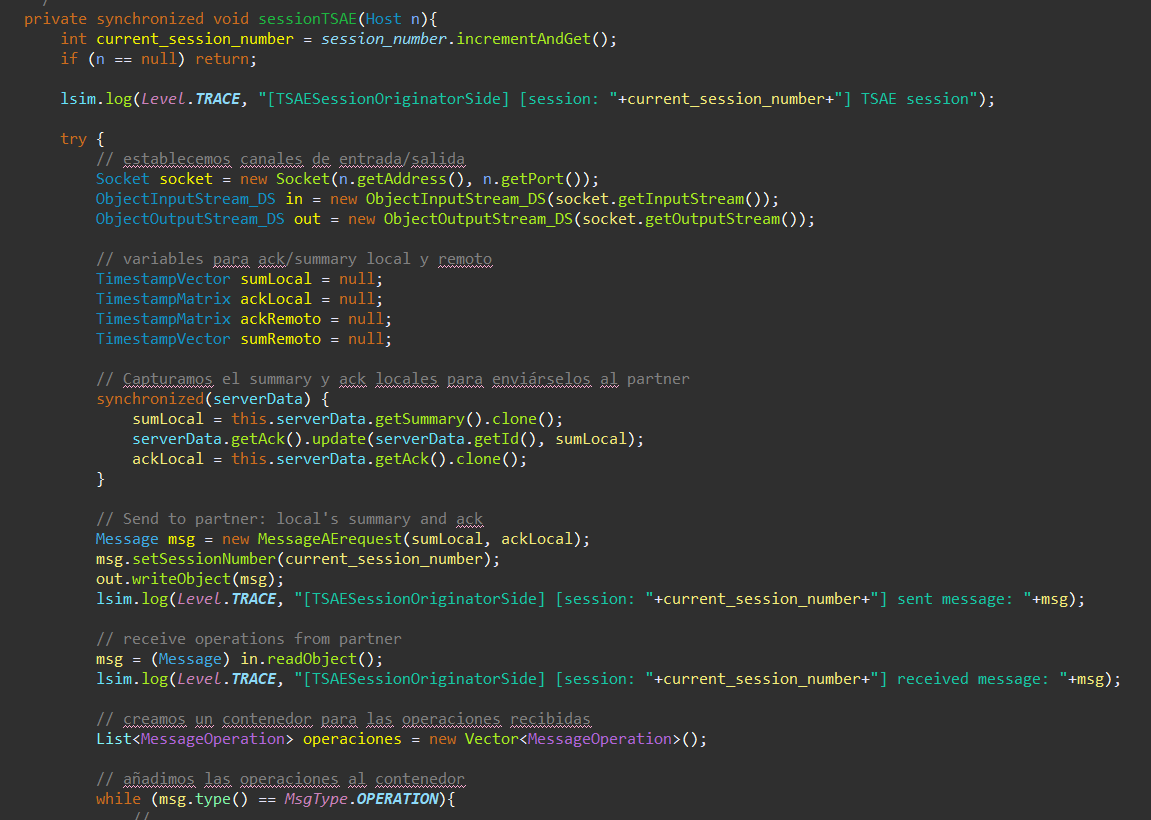
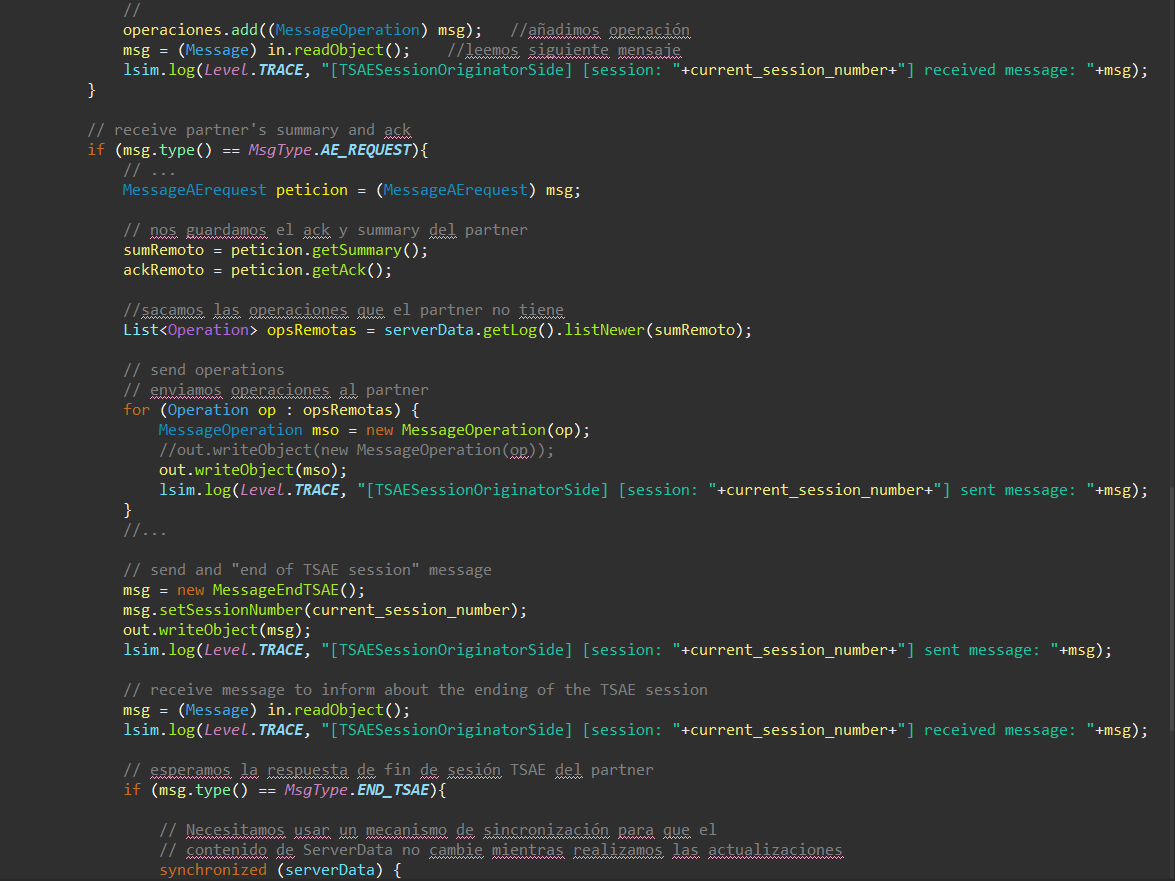
Con todo configurado, los grupos de lanzamiento aparecen en el menú de ejecución. Las pruebas locales se ejecutan con normalidad, y es posible *hacer debug* y acceder a la salida individual de todos los procesos.

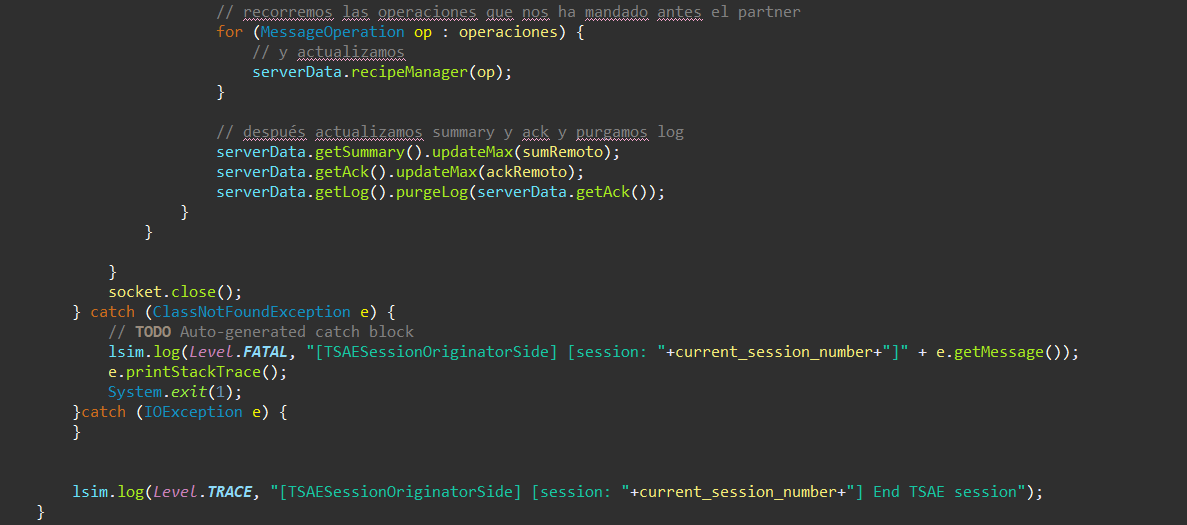


Ejecución de la fase 3 desde Eclipse. Se visualiza la salida del componente TestServer.

# Anexo 2: Métodos TSAESession\* sessionTSAE() y run()

* **TSAESessionOriginatorSide.sessionTSAE**:



* **TSAESessionPartnerSide.run():**

