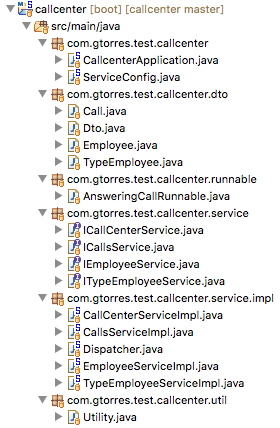
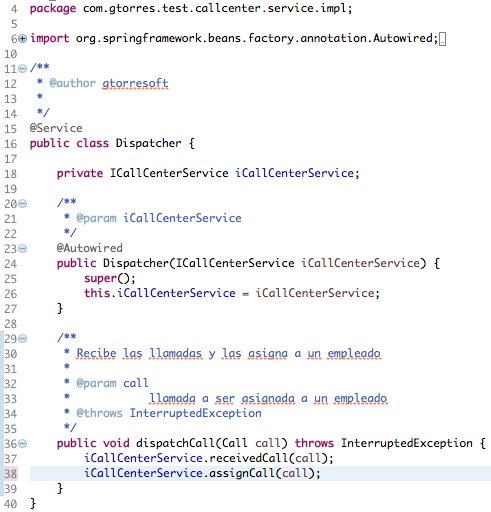
DESARROLLO EJERCICIO JAVA CALLCENTER



# REQUERIMIENTO 1

**Debe existir una clase Dispatcher encargada de manejar las llamadas, y debe contener el método dispatchCall para que las asigne a los empleados disponibles.**

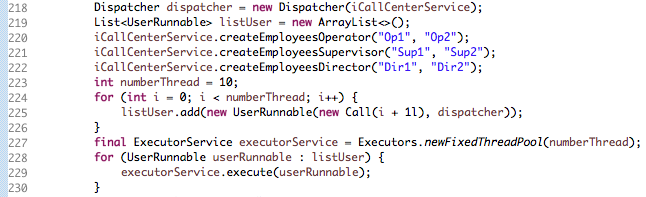


# REQUERIMIENTO 2 y 3

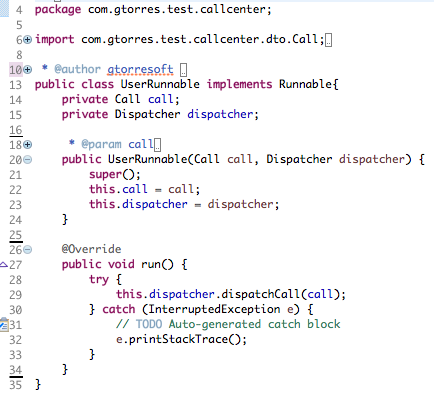
**Requerimiento 2: El método dispatchCall puede invocarse por varios hilos al mismo tiempo.**

**Requerimiento 3: La clase Dispatcher debe tener la capacidad de poder procesar 10 llamadas al mismo tiempo (de modo concurrente).**

La siguiente imagen muestra una porción de código que se repite en las pruebas unitarias en la clase **DispatcherTest.**



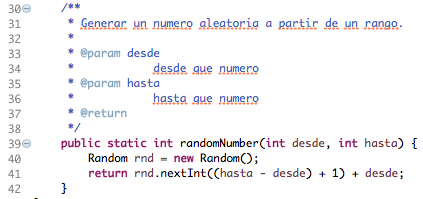
En la imagen anterior se evidencia la creación de una lista de 10 objetos **UserRunnable** que no son mas que 10 hilos para ejecutarse que simularan 10 llamadas concurrentes (Linea 224-226). Los hilos para su creación se les debe inyectar un objeto **Call** y uno **Dispatcher.** **Call** es la llamada que realizaría el Usuario y **Dispatcher** tiene un método **DispatcherCall** que se invocara concurremente por los hilos (Linea 225). En la línea 229 se evidencia la ejecución de los hilos concurrentemente. A continuación se muestra la clase **UserRunnable**



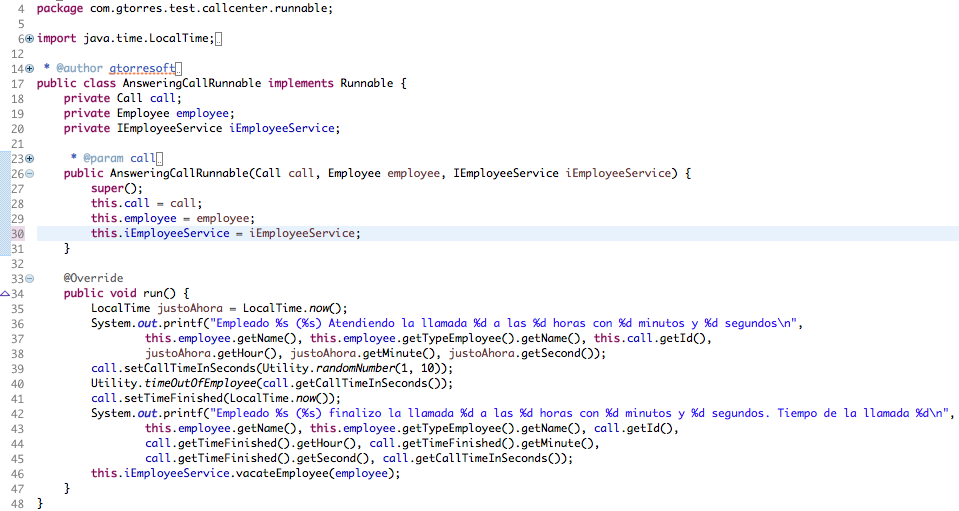
# REQUERIMIENTO 4

**Cada llamada puede durar un tiempo aleatorio entre 5 y 10 segundos.**

Se creo una clase Utility donde se crean métodos de utilidad general. En esta clase se crea el método **randomNumber** que retorna un numero aleatorio entre un rango determinado. A continuación se muestra el método.



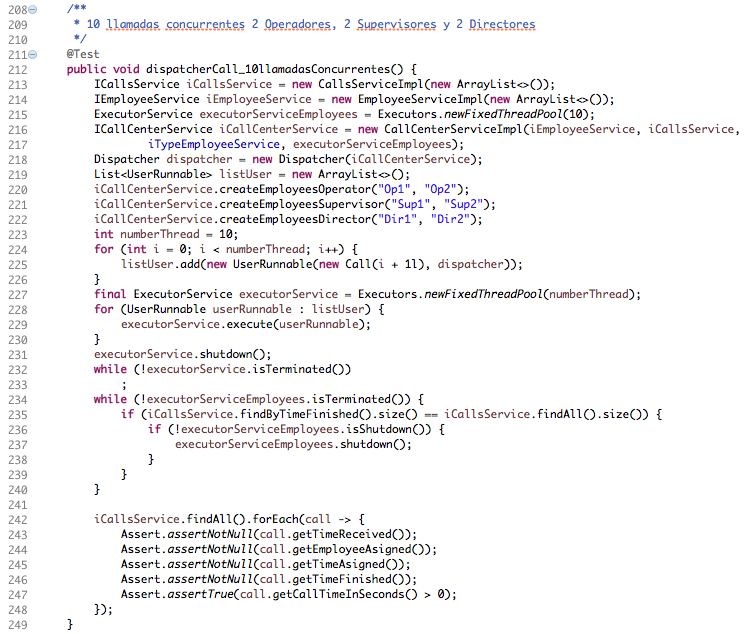
La asignación de este numero aleatorio se realiza en la ejecución de la atención de la llamada por un empleado. Para esto se creo una clase **AnsweringCallRunnable** que no es mas que un hilo que se abre para atender un llamada cuando un empleado esta disponible. A continuación la clase **AnsweringCallRunnable** en la línea 39 la asignación de este numero aleatorio**.**



# REQUERIMIENTO 5

**Debe tener un test unitario donde lleguen 10 llamadas.**

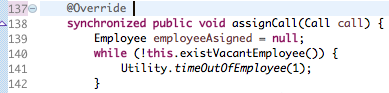
En la clase **DispatcherTest** se realizan test unitario para 2, 4, 5, 10 y 15 llamadas concurrentes donde se acierta que la atención de todas las llamadas hechas. A continuación el test de 10 llamadas concurrentes.



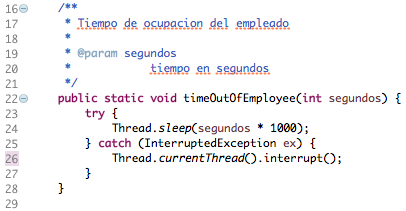
Extras/Plus 1

**Dar alguna solución sobre qué pasa con una llamada cuando no hay ningún empleado libre**

En el método ***CallCenterServiceImpl.assignCall(Call call)*** se valida que exista empleados desocupados; Si no es el caso, se hace una espera de 1 segundo preguntado si existe un empleado disponible, hasta que exista un empleado disponible no se hace la asignación de la llamada entrante.

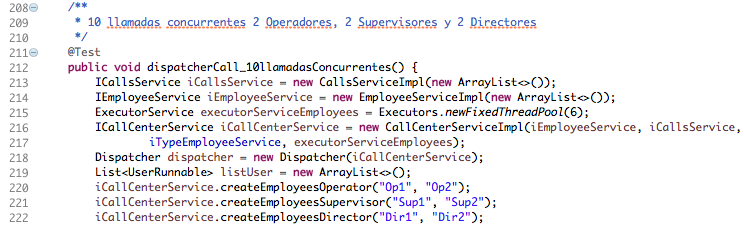


En la línea 141 se evidencia el llamada de un método ***Utility.timeOutOfEmployee*** que por un segundo duerme el hilo en ejecución. A continuación el método ***Utility.timeOutOfEmployee***



Extras/Plus 2

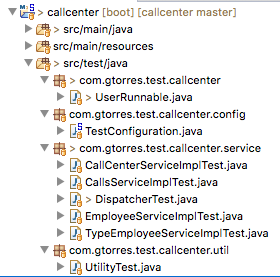
**Dar alguna solución sobre qué pasa con una llamada cuando entran más de 10 llamadas concurrentes.**



El uso la interface **ExecutorService** ayuda a crear un pool de hilos controlando la cantidad de estos a ejecutar concurrentemente. Si existen todos los hilos del pool en ejecución, este encola los hilos hasta que algún hilo del pool termine su ejecución. En los test unitarios en la línea 215 se evidencia la creación de un pool de máximo 6 hilos a ejecutar cocurrentemente. 6 hilos correspondientes a los 6 empleados que atenderán las llamadas. En las líneas 220-222 se evidencia la creación de los empleados 2 Operadores, 2 Supervisores y 2 Directores.

Extras/Plus 3

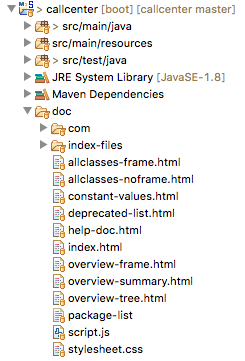
**Agregar los tests unitarios que se crean convenientes.**



Se hicieron las pruebas unitarios cumpliendo con un cubrimiento de mas de un 90% ya que es muy importante testear todos los métodos. Los test se ubican en la carpeta propia de Maven **src/test/java.**

Extras/Plus 4

**Agregar documentación de código**



La documentación del código puede evidenciarse dentro de la carpeta del proyecto en una carpeta llamada **doc.** Ver imagen anterior.