**Juan Carlos Eduardo Hidalgo**

**Solución de CallCenter para Almundo**

**Tecnologías Usadas**

1. Java 1.8.0\_171
2. Eclipse Java EE IDE for Web Developers. Versión: Oxygen. Release 4.7
3. Plugin para pruebas unitarias con JUnit

* junit-jupiter-api 5.4.1
* junit-jupiter-engine 5.4.1
* junit-platform-launcher 1.4.1

1. Plugin para pruebas de Cobertura

* jacoco-maven-plugin 0.8.3

1. Plugin de Maven 4.0

* maven-jar-plugin 3.1.0
* maven-compiler-plugin 3.8.0
* maven-surefire-plugin 2.22.1
* maven-dependency-plugin 3.1.1

1. Repositorio de Maven Central

* http://repo.maven.apache.org/maven2

**Desarrollo**

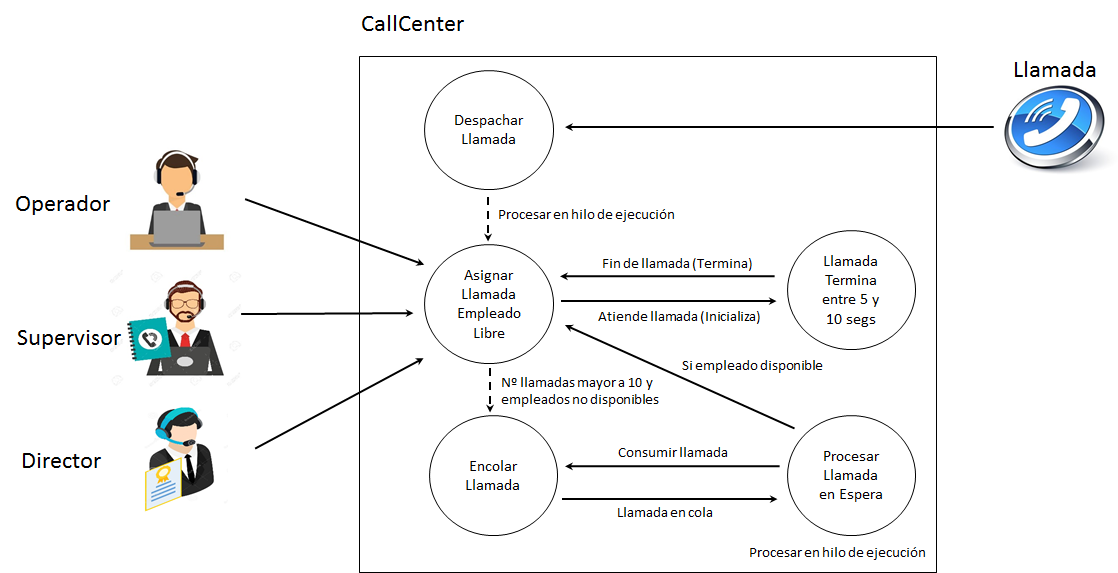
1. Proyecto CallCenter
2. Artefacto JAR: \target\CallCenter-0.0.1-SNAPSHOT.jar
3. Pruebas Unitarias: \src\test\java\callcenter\despachador\CallCenterUnitTest
4. Cobertura: \target\site\jacoco\index.html
5. Para el desarrollo se aplicaron los principios SOLID y buenas prácticas de POO. Se añadieron los tests unit necesarios y el código está completamente documentado: Clases, atributos, métodos y algoritmos.
6. Recompilar el proyecto ya que borre el directorio \target para no subir los binarios.

**Solución a Alto Nivel**

La solución general se encuentra reflejada en el diagrama que se anexa abajo. La idea central es manejar un CallCenter que tiene como capacidad máxima 10 empleados (incluyendo supervisor y director).

Cuando el número de llamadas llega máximo hasta 10 concurrentemente, cada empleado tiene la posibilidad de atender una llamada. Todas las llamadas se atienden en un hilo diferente. Si entran más llamadas de la capacidad máxima de empleados que es 10, se debe procesar una cola de llamadas en espera y atenderla en el instante que un empleado esté disponible. En este caso hay un hilo especial que atiende solamente llamadas en espera.

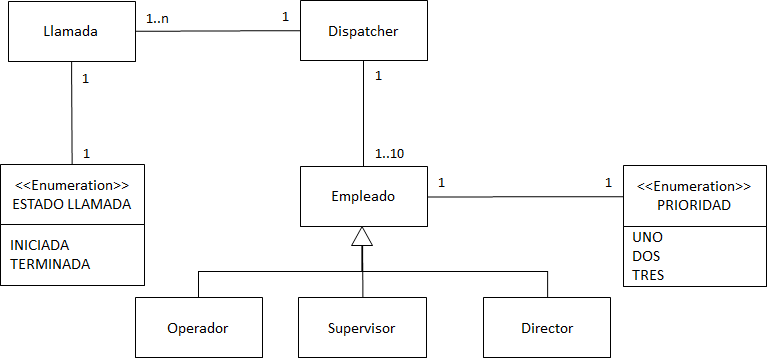
Las llamadas se demoran un tiempo aleatorio entre 5 y 10 segundos, es decir que en el peor caso cuando todos estén atendiendo una llamada, tardaría 100 segundos y en el mejor de los casos 50 segundos. Si hay llamadas en cola la tomaría el primer empleado que esté disponible.



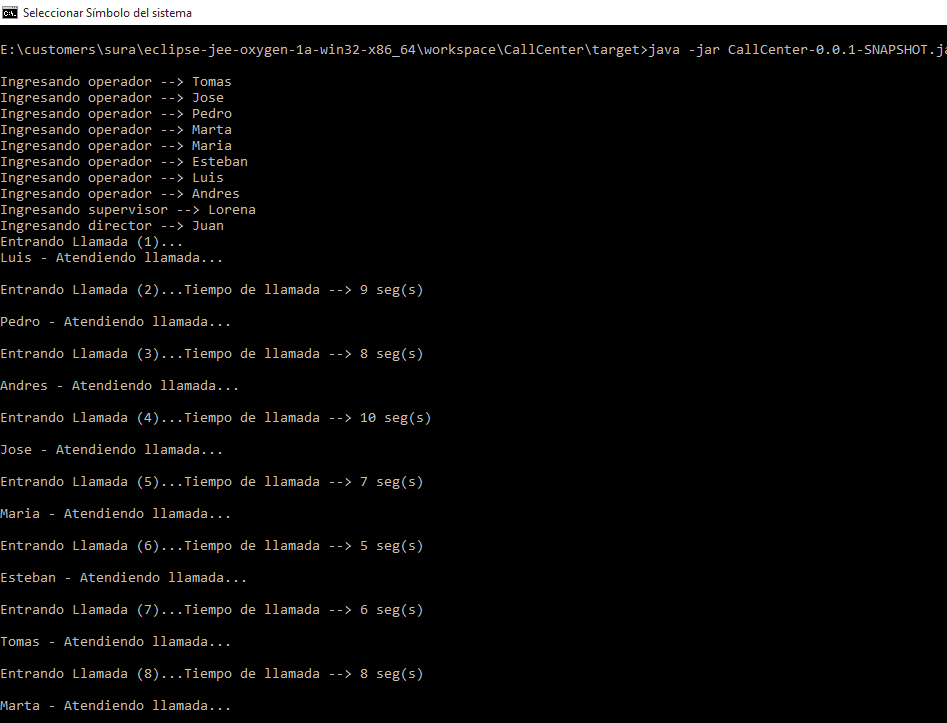
**Diagrama de Clases**

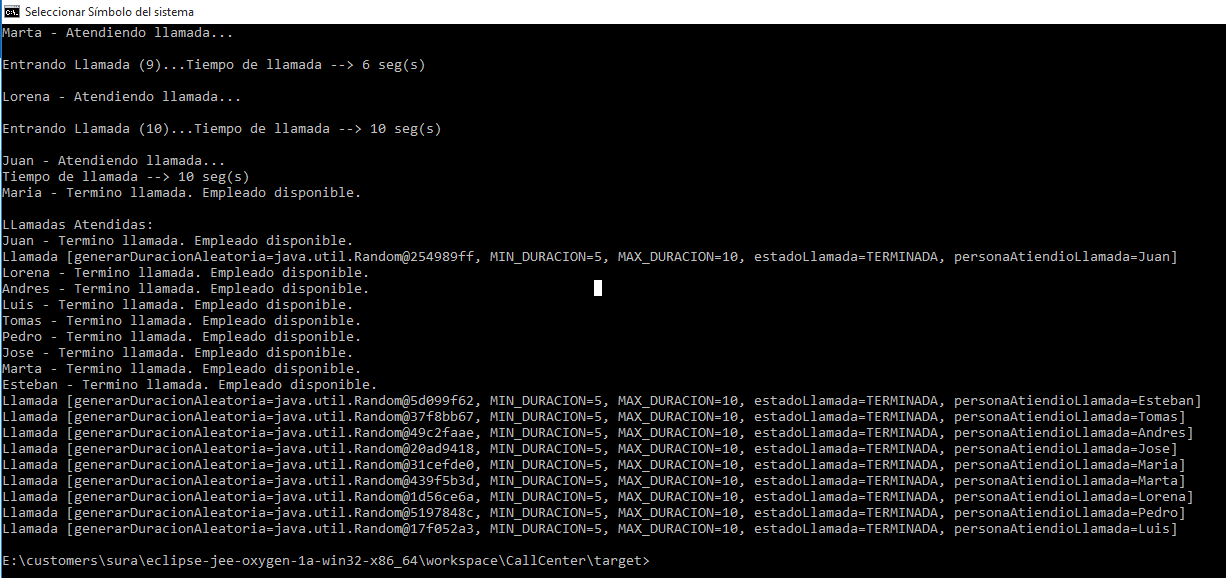
Su muestra un diagrama de clases asociado a la solución del problema, donde básicamente se determinan las clases involucradas:

* Clase Dispatcher: Clase que representa el despachador de llamadas
* Clase Llamada: Clase que representa la llamada entrante y que un empleado debe atender
* Enumerado Estado Llamada: Indica básicamente el estado de inicio de una llamada (INICIADA) y cuando la llamada finaliza (TERMINADA)
* Clase Empleado: Es una clase abstracta que representa en forma generalizada a un empleado
* Enumerado Prioridad: Indica la prioridad de un empleado para atender una llamada
* Clase Operador: Extiende de la clase abstracta Empleado, se le asigna una prioridad en la llamada que en este caso es 1
* Clase Supervisor: Extiende de la clase abstracta Empleado, se le asigna una prioridad en la llamada que en este caso es 2 cuando no hay operadores disponibles para atender una llamada
* Clase Director: Extiende de la clase abstracta Empleado, se le asigna una prioridad en la llamada que en este caso es 3 cuando no hay operadores ni el supervisor disponibles para atender una llamada



**Ejecución de Programa JAR**

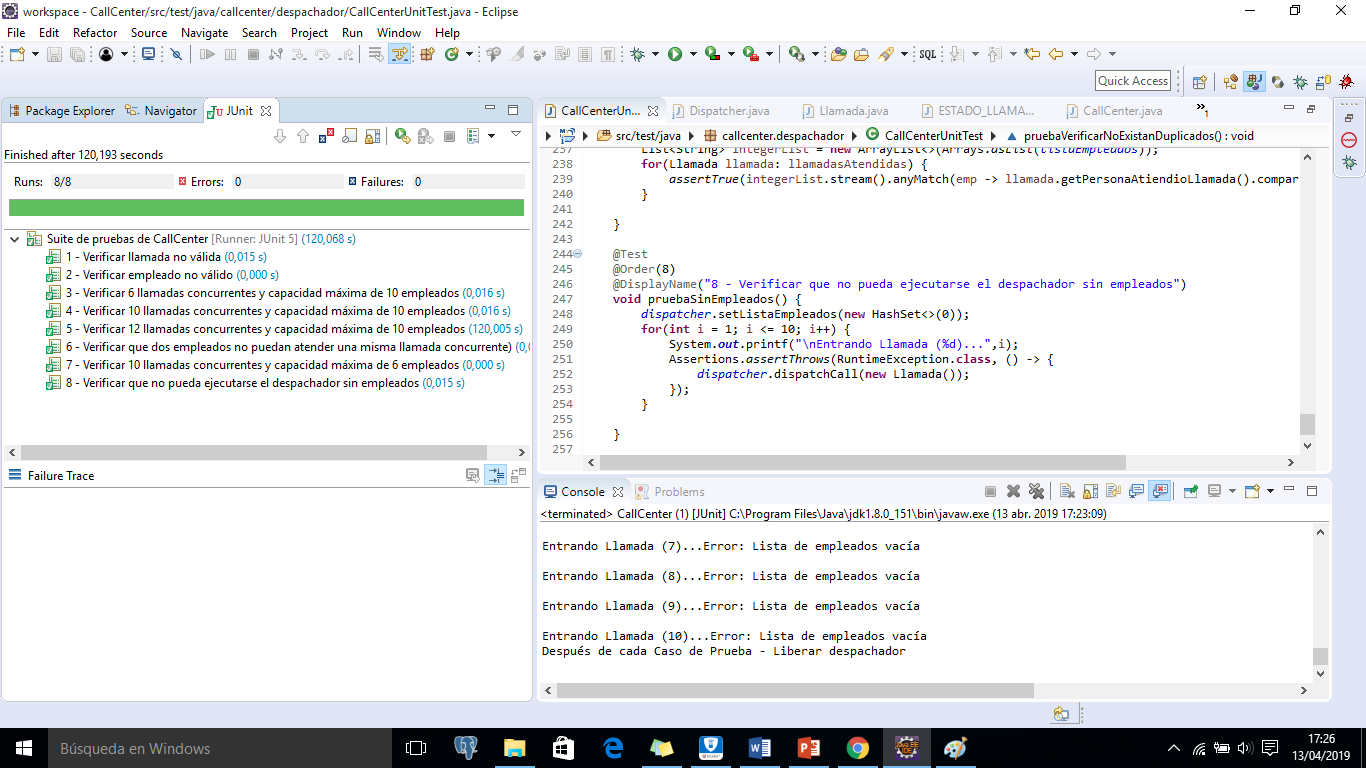




**Pruebas Unitarias**

La prueba unitaria solicitada es la asociada al número 4 donde se verifican 10 llamadas concurrentes con la capacidad máxima de 10 empleados. Sin embargo, en la práctica los desarrolladores necesitamos realizar pruebas que permitan la aumentar la cobertura, verificando las posibles fallas que se puedan presentar al momento de implementar un método o función. Brevemente se resume cada una de las pruebas:

1. Verificar llamada no válida: Si el objeto es null el método dispatchCall debe controlar la excepción.
2. Verificar empleado no válido: Al momento de asignar el empleado al despachador si el objeto es null también debe manejarse para evitar fallas.
3. Verificar 6 llamadas concurrentes y capacidad máxima de 10 empleados: Menos llamadas con la capacidad máxima de empleados
4. Verificar 10 llamadas concurrentes y capacidad máxima de 10 empleados: Esta prueba que cada empleado atiende una llamada (fue la solicitada).
5. Verificar 12 llamadas concurrentes y capacidad máxima de 10 empleados: Prueba donde el número de llamadas supera a la capacidad máxima y en ese caso pasaría a una cola de espera. Al momento que un empleado esté disponible atienda la que está en espera.
6. Verificar que dos empleados no puedan atender una misma llamada concurrente: Si en la lista de llamadas atendidas aparece un empleado dos (2) veces es un error, ya que indica que una persona al mismo tiempo atendió dos llamadas, lo cual no es muy práctico.
7. Verificar 10 llamadas concurrentes y capacidad máxima de 6 empleados: Supongamos que no asistieron todos los empleados al CallCenter, en este caso se verifica el comportamiento de las llamadas.
8. Verificar que no pueda ejecutarse el despachador sin empleados: Sin empleados las llamadas no pueden ser procesadas, hay que manejar este caso borde.



**Cobertura**

Se añadió cobertura de código que en pruebas unitarias para medir el grado en que el código fuente ha sido comprobado, y que permita determinar las partes críticas del código que no han sido comprobadas y las partes que ya lo fueron. Para este desarrollo la cobertura fue de un 97% (Ver imagen anexa abajo). Cada empresa determina el porcentaje mínimo de cobertura que los proyectos de software deben tener, en mi experiencia la mayoría tiene un 75%.

Ruta del archive de cobertura → CallCenter\target\site\jacoco\index.html

