Documentación Callcenter:

El sistema de callcenter solicitado por el ejercicio se implementó utilizando Java 8, maven 3.3.9 y JUnit 4.12.
Para construir el archivo .jar ejecutar el siguiente comando:

```
mvn package -Dmaven.test.skip=true
```

Eliminar el parámetro -Dmaven.test.skip=true si se quieren ejecutar los test del sistema al construirlo.
Ejecutarlo con el siguiente comando:

```
java -cp target\callcenter-0.0.1-SNAPSHOT.jar com.germanrobledo.callcenter.Callcenter
```

Diseño del sistema:

Se diseñó una clase Dispatcher que se encarga de manejar las llamadas mediante el método dispatchCall() (como fue solicitado en el enunciado). Para esta tarea la clase Dispatcher contiene un ExecutorService el cual se encarga de ejecutar cada llamada en un thread distinto y administra un pool de 10 threads (configurados por la constante MAX_THREADS de la clase Dispatcher.
Cuando se intenta ejecutar más de 10 llamadas en formar simultáneas estas son encoladas hasta que se libere un thread, Esto da solución al punto Extra/Plus “Dar alguna solución sobre qué pasa cuando entran mas de 10 llamadas concurrentes”.
El ExecutorService Se inicia en el constructor de la clase Dispatcher y se detiene mediante el método stop(). Si algún thread aún se está ejecutando este método espera SECONDS_TO_SHUTDOWN (constante de la clase Dispatcher) a que finalicen, si no termina en ese tiempo lo detiene.
La Clase Dispatcher contiene otro atributo que es la cola de operadores de llamada (los operadores de llamada son los operadores, supervisores y directores). Esta cola está modelada por la clase CallHandlerQueue y los operadores por CallHandler (especificadas más adelante).
El método dispatchCall() recibe por parámetro un número entero que es la cantidad de segundos que dura la llamada. La llamada es modelada por la clase Call la cual implementa la interface Runnable de manera que pueda ser ejecutada por un thread. dispatchCall() crea una instancia de Call pasandole como parametros al constructor la cola de operadores y los segundos de duración y luego submitea en el ExecutorService que se encarga de ejecutarla en alguno de su threads.

La clase CallHandler modela los operadores de llamadas, contiene dos atributos, el nombre del operador y su tipo. El atributo tipo modela mediante un Enum (ClassHandlerEnum) si es un operador, un supervisor o un director.
La clase CallHandlerQueue modela la cola de donde se van obteniendo los operadores de llamada. Esta clase contiene un LinkedBlockingDeque la cual bloquea el thread que ejecuta la call cuando esta vacía hasta que algún operador de llamada esté disponible, esto soluciona el punto Extra/Plus “Dar una solución sobre qué pasa con una llamada cuando no hay ningún empleado libre”. CallHandlerQueue implementa su propio método de inserción en la cola put() para cumplir con el enunciado que primero atienden los operadores, si no hay ninguno disponible un supervisor , y cuando no quedan supervisores libres los directores. Para cumplir con este enunciado el método put() inserta los operadores de llamada de manera que la cola siempre quede ordenada con los operadores primero los supervisores luego y los directores al final.
La lógica que utiliza es la siguiente:

Si es de tipo operador lo inserta primero en la cola.

Si es de tipo director lo inserta último en la cola.

Si es de tipo supervisor, si hay operadores los saca de la cola inserta el supervisor y luego vuelve a insertar todos los operadores en el orden que estaban.

La clase Callcenter contiene el método main de ejecución del sistema. Este ejecuta un ejemplo donde crea la cola de operadores , le inserta los operadores , crea el dispatcher y luego genera una cantidad al azar de llamadas (con duraciones al azar entre 5 y 10 seg) luego espera unos segundos y vuelve a generar llamadas. Esto lo realiza 3 veces. Luego espera durante 10 segundos para que terminen todas las llamadas pendientes y finalmente detiene al dispatcher.

Diagrama de Clases:

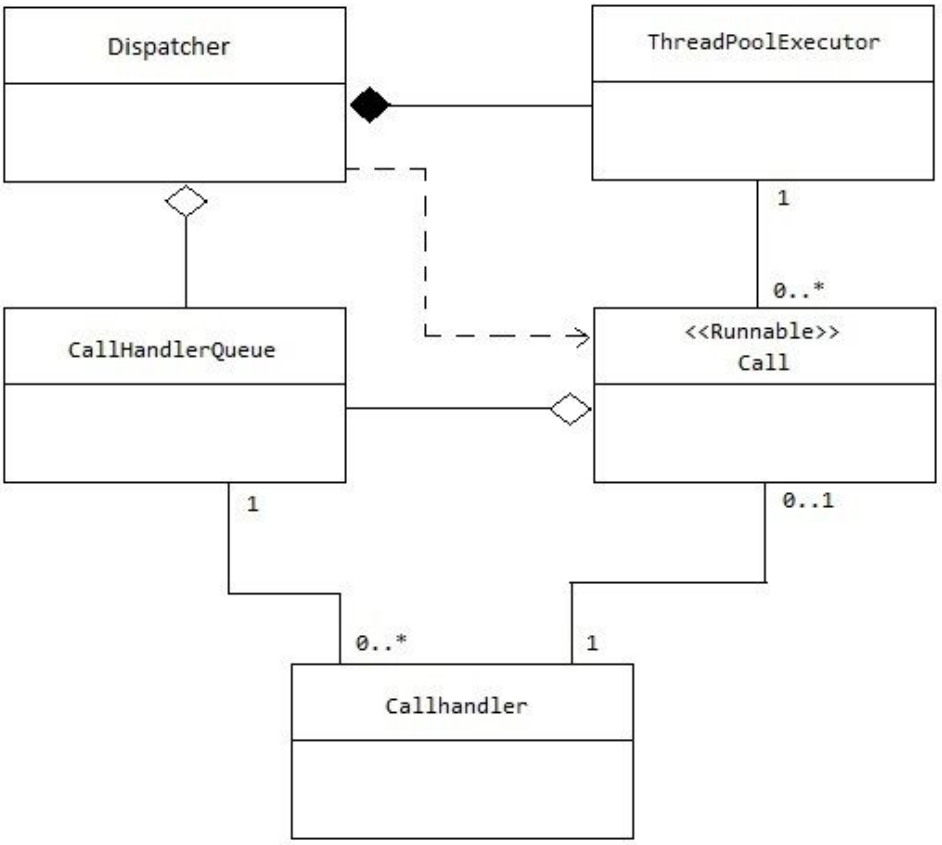
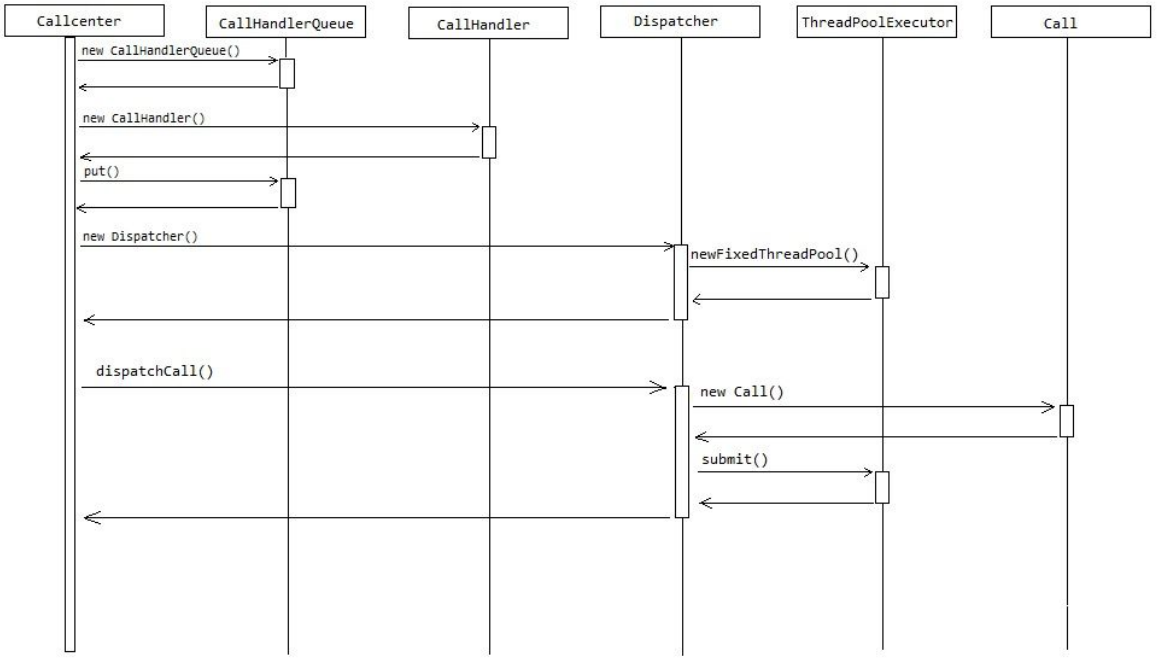


Diagrama de Secuencia:

Este diagrama modela el caso en el que el callcenter crea la cola de operadores el dispatcher y realiza una llamada.



Casos de Pruebas:

En la clase CallcenterTest se encuentran los test unitarios. Esta clase contiene un método startCallcenter() el cual crea la cola de operadores con 6 operadores e inicializa el dispatcher que se ejecuta antes de cada test unitario. También contiene un método shutdownCallcenter() el cual espera un tiempo definido por el atributo secondsToShutdown y luego detiene el dispatcher. Este método se ejecuta cuando termina cada test unitario.

Los test unitarios son los siguientes:

tenThreadsTest(): Ejecuta 10 llamadas en formas simultánea, como se solicitó en el enunciado.

moreCallsThanThreadsTest: Ejecuta 12 llamadas en forma simultánea, el Dispatcher solo puede ejecutar 10 threads al mismo tiempo, por lo cual este Test muestra cómo el sistema bloquea las llamadas cuando no hay threads disponibles hasta que alguno se libere.

moreCallsThanHandlers: Ejecuta 15 llamadas simultáneas con solo 6 operadores, este Test muestra el comportamiento del sistema cuando no hay operadores disponibles. Cómo bloquea las llamadas hasta que un operador es liberado.