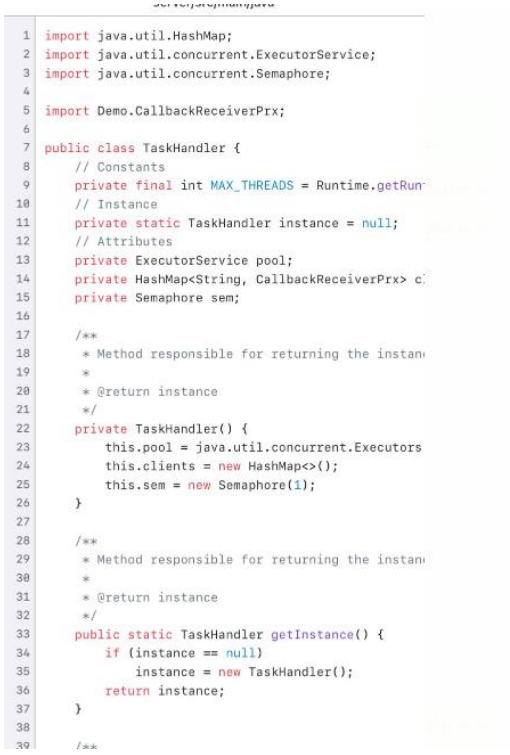
Análisis de saturación del servidor TimeOutException

Una vez se implementó el callback con la concurrencia, obtuvimos los siguientes resultados:

Numero	Maquina	Tiempo (ms)
1^3	Hgrid4	320
1^4	Hgrid5	493
1^5	Hgrid6	541
1^6	Hgrid7	978
1^7	Hgrid4	1291
1^8	Hgrid8	3256

En este caso se puede observar que aplicando una concurrencia real, donde cada procesador se encarga de realizar una tarea, lo que permite que los clientes se queden esperando una respuesta indefinida por parte del servidor. Gracias a esto se mejora la velocidad de respuesta del servidor a cada cliente. Finalmente se concluyó que no existe un límite (razonable) el cual llegue a colgar el sistema, debido a que un valor demasiado grande, los resultados empiezan a ser incongruentes y sin sentido

Comprobación de la concurrencia.



```
import java.util.HashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Semaphore;

import Demo.CallbackReceiverPrx;

public class TaskHandler {
    // Constants
    private final int MAX_THREADS = Runtime.getRun
    // Instance
    private static TaskHandler instance = null;
    // Attributes
    private ExecutorService pool;
    private HashMap<String, CallbackReceiverPrx> c
    private Semaphore sem;

    /**
     * Method responsible for returning the instan
     *
     * @return instance
     */
    private TaskHandler() {
        this.pool = java.util.concurrent.Executors
        this.clients = new HashMap<>();
        this.sem = new Semaphore(1);
    }

    /**
     * Method responsible for returning the instan
     *
     * @return instance
     */
    public static TaskHandler getInstance() {
        if (instance == null)
            instance = new TaskHandler();
        return instance;
    }

    /**
```