Entregable 2 – María Agustina Díaz y Miranda Martinez

Desiciones de diseño tomadas:

El sistema maneja pedidos concurrentemente utilizando un *ExecutorService* con un *FixedThreadPool*. Esto significa que se cuenta con un grupo fijo de hilos para procesar todas las tareas de forma concurrente, manteniendo el uso de recursos controlado. Los pedidos se procesan en base a su prioridad usando una cola (*PriorityBlockingQueue*), donde los pedidos urgentes se atienden antes que los normales.

El procesamiento de cada pedido se divide en tres etapas: procesar el pago, empaquetar, y enviar. Cada etapa tiene su propio semáforo para limitar la cantidad de hilos que pueden ejecutarse simultáneamente:

- Pago: 3 hilos
- Empaquetado: 4 hilos
- Envío: 3 hilos

Se realizaron pruebas manuales asignando diferentes cantidades de hilos y valores de semáforos. La configuración seleccionada es la que reduce el tiempo total de procesamiento, respetando la carga de cada tarea y garantizando la eficiencia.

Los pedidos urgentes tienen prioridad y se insertan primero en la cola. Esto se hace con una *PriorityBlockingQueue*, que utiliza el campo esUrgente para decidir el orden de procesamiento.

Inicialmente se consideró usar *CachedThreadPool* en lugar de *FixedThreadPool*, que ajusta dinámicamente el número de hilos según la carga. Sin embargo, esta opción no fue adecuada debido a la falta de control sobre el número máximo de hilos, lo que podría causar sobrecarga en el sistema y tambien la opción fija nos permite manejar las cargas sabiendo cuan pesada es cada etapa.

Pruebas unitarias

Las pruebas unitarias verifican que los pedidos cambian correctamente de estado (PROCESANDO_PAGO, EMPAQUETANDO, ENVIANDO). Usando JUnit, se testean tres aspectos como agregar pedidos, procesar pedidos y cerrar el sistema.

Elección de ForkJoinPool

Se eligió ForkJoinPool sobre los streams paralelos porque proporciona un mayor control sobre el procesamiento paralelo de los pedidos, optimiza el uso de los recursos del sistema para manejar tareas ligeras y garantiza una coordinación más efectiva mediante semáforos. Esto asegura que podamos procesar los pedidos de manera eficiente y con la

flexibilidad necesaria para ajustar el paralelismo a las características del sistema. Además, al agregar esta funcionalidad en última instancia, pudimos notar una gran mejora en el tiempo de ejecución del programa.

En conclusión, el sistema está diseñado para manejar 100 pedidos concurrentes eficientemente, usando ExecutorServicecon un FixedThreadPool para controlar los hilos y una PriorityBlockingQueue para priorizar pedidos urgentes. Los semáforos garantizan un acceso seguro a los recursos en cada etapa (pago, empaquetado, envío), evitando conflictos. Además, el diseño modular y las pruebas unitarias aseguran estabilidad y buenas prácticas en concurrencia, cumpliendo con los objetivos de rendimiento y seguridad de acceso en Java.