# Entregable 2 – Concurrencia

La empresa tiene 3 grandes procesos, el procesamiento de pago, el empaquetado de pedidos y el envio los cuales se realizan por lotes de trabajo, estas se tienen que dar en ese orden, una vez que sean pagas, se procede a preparar el pedido y luego se le imprime la etiqueta. Entonces una tiene que esperar a que la anterior suceda. Por lotes de trabajo, nos referimos a que en una ventana de tiempo, llegan una cantidad X de pedidos y se trabaja con ellos.

A su vez, esta empresa maneja un sistema de prioridades, lo que marca que aquellos paquetes con mayor prioridad, tienen que pasar antes. Pero que significa esto para nosotros? Para empezar hay que ver cada uno de los procesos. Si nos centramos en el procesamiento de pago, este no puede ser interumpido para poder realizar el de mayor prioridad, sino que espera a que termine el proceso actual y se produce el proximo.

Por otra parte esta el empaquetado, como esto permite trabajar en paralelo no hay mucho drama por hacer, ni bien llegue uno con prioridad mas alta que algun paquete, este va primero. Y para el evio es un poc mas de lo mismo.

Para simular el proceso de los pedidos, ya sea el pago, el empaquetamiento o el envio, usamos el Thread.sleep(1000) para marcar una parada de tiempo, asi se puede parar el proceso por un rato y simular que se estan haciendo cosas.

Vamos a tener 5 clases, la primera esta destinada a definir a los pedidos e implementar un id y un boolean para marcar si es urgente. Luego vienen 3 clases destinadas a los 3 procesos donde van a simular hacer las distintas tareas. La clase importante es la 5ta, donde se va a encontrar la logica de esto. Esta clase va a ser SistemaPedidos

## Desiciones de codigo

Para poder hacer este proceso, se me presentaron 2 opciones, una que maneje una sola queue y un solo executor que haga de forma serial todos los pasos, la otra opcion fue paralelizar los pasos y que tenga el pool de hilos divididos entre cada uno de pasos. En un principio pense que iba a ser mas rapido ejecutar todo de forma serial, pero luego cuando integre la prioridad, era mas esencial sacar primero los pedidos de mayor prioridad, para esto el paralelismo es la forma de hacerlo ya que mientras unos hilos procesan el pago, otros se encargan de hacer el empaquetado y otros los envios.

Entonces luego de hacer ambos codigos, ya que teniamos un cronometro para medir el tiempo total del proceso, ambos codigos con misma cantidad total de hilos y misma cantidad de pedidos. Los resultados no solo fueron que los pedidos de mayor prioridad se despachaban antes en el codigo paralelo, sino que tambien la produccion total fue mas eficaz. Por eso hoy decido presentar una solucion donde los procesos estan paralelizados.

Por lo tanto, el codigo que se entrega esta en la carpeta de codigo principal, mientras que la comparacion se encuentra en codigo secundario

## Estructuras adicionales

Para implementar el ForkJoinPool, existe una estructura llamada CompletableFuture con metodos async lo que permite evitar que unos procesos se lleven a cabo antes que otros. A su vez permite cerrar el sistema con todos los procesos terminados.

Bio: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html>

Para manejar las prioridades se va a tener una PriorityBlockingQueue, esta va a ordenar los pedidos del lote segun su prioridad, asegurando que estos salga primeros. Estos van a tener dentro de su clase un atributo que representa esta prioridad.

Bio: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/PriorityBlockingQueue.html