**Steam:** es una secuencia de elementos que soporta diferentes tipos de operaciones para procesar datos de manera declarativa y funcional. Introducido en java 8 permite realizar operaciones de mapeo filtrado y reducción de manera más eficiente. No almacenan datos, permite escribir un código mas limpio y legible para operaciones complejas.

**Los Parallel Stream** es un conjunto de operaciones aplicables a listas permiten procesar grandes volúmenes de datos de manera eficiente utilizando múltiples hilos.

*En resumen, tanto los hilos como los Parallel Streams ejecutan tareas en paralelo, pero los hilos brindan un control más directo y detallado sobre la concurrencia y la sincronización, mientras que los Parallel Streams ofrecen una abstracción de más alto nivel y una interfaz más fácil de usar para el procesamiento paralelo de datos en colecciones.*

**Estrategias de diseño**

**Descomposición en tareas:** Dividir el trabajo en tareas independientes, esta técnica mejora la eficiencia, escalabilidad y mantenibilidad de la aplicación.

**Descomposición en datos:** Dividir los datos en bloques procesables en paralelo, cada bloque puede ser procesado en paralelo, lo que mejora el rendimiento y la eficiencia de la aplicación.

**Descomposición funcional**: Dividir el trabajo en funciones o módulos, cada módulo se encarga de una parte específica del problema, y estos módulos pueden ejecutarse en paralelo, mejor.

**Patrones de diseño**

Los patrones de diseño son soluciones generales y reutilizables a problemas recurrentes en el diseño de software. Estos patrones representan mejores prácticas que los desarrolladores pueden aplicar para resolver problemas específicos dentro de un contexto dado.

**Características**

Abstracción: Proporcionan una solución abstracta que se puede adaptar a múltiples situaciones.

Reutilización: Facilitan la reutilización de soluciones probadas y optimizadas.

Estandarización: Promueven la consistencia y estandarización en el desarrollo de software.

Documentación: Generalmente están bien documentados, lo que facilita su comprensión e implementación.

**Los puntos de sincronización** son lugares en el programa donde se coordina la ejecución de múltiples hilos o procesos para asegurar que ciertas condiciones se cumplan antes de continuar.

**Herramientas y Técnicas de Sincronización**

Mutex (Mutual Exclusion): Proporciona exclusión mutua, asegurando que solo un hilo a la vez pueda acceder a una sección crítica.

Semáforos: Utilizados para controlar el acceso a recursos compartidos mediante contadores.

Barreras: Utilizadas para sincronizar hilos en diferentes etapas de ejecución.

Monitores: Abstracciones que combinan mutexes y variables de condición para facilitar la sincronización.

**¿Cuáles son las características de los patrones de diseño?**

Los patrones de diseño se utilizan para optimizar la manera en que se desarrolla un software en un entorno concurrente. Se basa en la división de tareas según la utilidad y necesidad de las mismas.

**Menciones las ventajas del patrón Productor-Consumidor.**

la división de tareas donde el productor es el que se encarga de generar los recursos que el consumidor utilizara para el desempeño de las tareas que se precisen llevar a cabo.

**Patrón de Trabajador (Worker)**

Distribuye las tareas entre un conjunto de hilos de trabajadores que ejecutan las tareas concurrentemente.

Simplifica la gestión de múltiples tareas y permite un balanceo de carga eficiente.

**Patrón Pipeline**

Divide el procesamiento en una serie de etapas, donde cada etapa puede ejecutarse en paralelo.

Mejora la estructura del procesamiento en etapas secuenciales y Permite el procesamiento paralelo y la reducción del tiempo total de procesamiento.

**Patrón MapReduce**

Divide el procesamiento en dos fases: Map, donde se aplica una función a cada elemento de un conjunto de datos, y Reduce, donde se agregan los resultados.

Escalabilidad y eficiencia en el procesamiento de grandes datos y facilita la paralelización de tareas distribuidas.

**ConcurrentHashMap** es una clase en Java que forma parte del paquete java.util.concurrent y proporciona una implementación de un mapa (map) que permite operaciones concurrentes seguras y eficientes. Está diseñada para ser utilizada en entornos multihilo (multithreading) donde múltiples hilos pueden acceder y modificar el mapa simultáneamente sin causar inconsistencias o condiciones de carrera.

**CopyOnWriteArrayList** es una implementación de la interfaz List en Java, que forma parte del paquete java.util.concurrent. Está diseñada para manejar escenarios de concurrencia en los que las operaciones de lectura son mucho más frecuentes que las operaciones de escritura. Su principal característica es que crea una copia de la matriz (array) subyacente cada vez que se realiza una operación de modificación (como add, set, remove). Esta característica garantiza que las operaciones de lectura (como get y iterator) no necesitan bloqueos y son seguras para ser ejecutadas concurrentemente.