Introducción a hilos en java

Un hilo (o *thread*) es la unidad más pequeña de procesamiento dentro de un proceso. Java ofrece soporte nativo para la programación concurrente mediante la clase `Thread` y la interfaz `Runnable`. Los hilos permiten ejecutar varias tareas simultáneamente dentro de una misma aplicación, aprovechando los recursos del procesador y mejorando la eficiencia en tareas complejas como cálculos paralelos, procesamiento de archivos, o comunicación en red.

Existen dos formas principales de crear hilos: heredando de la clase `Thread` o implementando la interfaz `Runnable`. La herencia de `Thread` permite sobrescribir el método `run()` para definir la lógica del hilo. Por otro lado, la interfaz `Runnable` se implementa cuando se desea pasar la tarea a un objeto de tipo `Thread`. Esta última opción es preferida en entornos complejos donde se requiere herencia múltiple o reutilización del código de tarea.

```
public class MiHilo extends Thread {
	@Override
	public void run() {
	System.out.println("Hilo ejecutándose");
	}
}

public class MiRunnable implements Runnable {
	@Override
	public void run() {
	System.out.println("Hilo usando Runnable");
	}
}
```

Java proporciona métodos como `start()`, `join()`, `sleep()` y `interrupt()` para gestionar los hilos. El método `start()` inicia un hilo, permitiendo que el método `run()` se ejecute en paralelo con el hilo principal. Con `join()`, un hilo espera a que otro termine su ejecución antes de continuar. Además, `sleep()` pausa un hilo durante un tiempo específico, y `interrupt()` permite interrumpir un hilo en ejecución, manejando tareas que pueden necesitar ser detenidas de manera controlada.

Cuando varios hilos acceden a recursos compartidos, pueden ocurrir condiciones de carrera o resultados inconsistentes. Para evitar estos problemas, Java ofrece mecanismos de sincronización como la palabra clave `synchronized`, que permite bloquear un método o un bloque de código para que solo un hilo pueda acceder a él a la vez. Adicionalmente, existen clases más avanzadas como `ReentrantLock` y estructuras concurrentes de las bibliotecas del paquete `java.util.concurrent` para gestionar concurrencia de forma segura y eficiente.

```
public synchronized void metodoCritico() {
```

```
        // Código que solo un hilo puede ejecutar a la vez
}
```

En aplicaciones que requieren ejecutar múltiples tareas concurrentes, es recomendable utilizar *thread pools* para gestionar los hilos de manera eficiente. El uso de pools evita la creación excesiva de hilos, lo que puede saturar la CPU y la memoria. Java ofrece la clase `ExecutorService` para gestionar estos pools. Un pool de hilos permite reutilizar hilos ya existentes, reduciendo la sobrecarga de creación y finalización de hilos, lo que es esencial en servidores y aplicaciones de alto rendimiento.

```
ExecutorService executor = Executors.newFixedThreadPool(5);
executor.submit(() -> {
        System.out.println("Tarea ejecutada por un hilo del pool");
});
executor.shutdown();
```

Este enfoque del manejo de hilos en Java permite una gestión eficiente de la concurrencia, mejorando la capacidad de las aplicaciones para realizar múltiples tareas al mismo tiempo.