CONCURRENCY CONCEPTS AND CONCURRENT COLLECTIONS IN JAVA

Sandra Kumi

REVIEWER Mr. Thomas Darko

Introduction to Concurrency

Concurrency in computing refers to the execution of multiple instruction sequences at the same time. This can occur on a single processor or across multiple processors.

Multithreading is one way to achieve concurrency, where multiple threads within a process are executed in a way that may seem to overlap in time, creating the illusion of simultaneous execution. However, it's essential to understand the differences between multithreading and concurrency, and the challenges that come with concurrent programming.

Understanding Threads in Java

In Java, threads are the smallest unit of execution. Java provides two main ways to create a thread: by extending the Thread class or by implementing the Runnable interface.

Threads have a lifecycle that includes states like NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, and TERMINATED. However, multithreading introduces challenges like race conditions, where multiple threads modify shared data simultaneously, leading to inconsistent states. These issues can be mitigated through synchronization.

Synchronization and Thread Safety

Synchronization ensures that only one thread can access a critical section of code at a time, preventing race conditions.

Java provides synchronized blocks and methods to achieve thread safety. However, synchronization must be used cautiously to avoid deadlocks, where two or more threads are waiting indefinitely for each other to release resources.

Java Memory Model

The Java Memory Model defines how threads interact through memory and what behaviors are allowed in concurrent programs. It specifies rules for visibility and atomicity, ensuring that actions in one thread are visible to others.

The 'happens-before' relationship is a key concept, indicating that memory writes by one specific statement are visible to another specific statement. Synchronization constructs and volatile variables are tools provided by Java to manage visibility.

Introduction to Concurrent Collections

Concurrent collections are part of the Java Collections Framework and provide thread-safe operations on collections.

Unlike traditional collections like ArrayList or HashMap, which require external synchronization, concurrent collections are designed to handle concurrent access with minimal contention.

Key Concurrent Collections in Java

Java provides several concurrent collections, each optimized for specific scenarios:

1. **ConcurrentHashMap**: A thread-safe variant of HashMap that allows safe concurrent accesswithout locking the entire map.
2. **CopyOnWriteArrayList**: Suitable for scenarios with frequent reads and infrequent writes. Theentire array is copied on each write operation.

3. **ConcurrentLinkedQueue**: A non-blocking queue that is ideal for producer-consumer scenarios.

Comparing Concurrent and Non-Concurrent Collections

Concurrent collections generally offer better performance in multithreaded environments compared to non-concurrent collections that are synchronized externally.

However, concurrent collections might have higher memory overhead and can be more complex to use. The choice between concurrent and non-concurrent collections depends on the specific needs of your application.

Best Practices for Concurrency in Java

To write effective concurrent programs in Java, consider the following best practices:

- Avoid shared mutable state whenever possible.

- Use concurrent collections for shared data access.

- Prefer higher-level abstractions like Executors and Thread Pools over manually managing threads.

- Be mindful of synchronization to avoid deadlocks and performance bottlenecks.