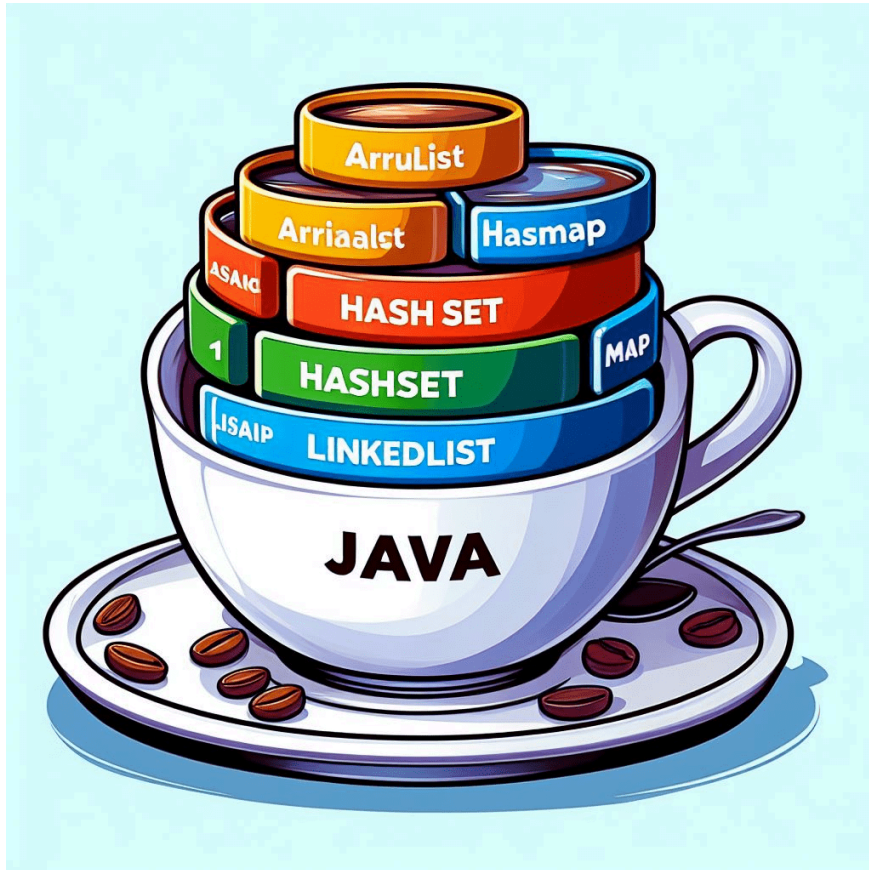


Java Collections and Their Thread-Safe Versions



Java Collections

As a Java developer, understanding collections is essential. Collections provide a way to store, manipulate, and retrieve data efficiently. In this article, we'll explore various Java collections and discuss their thread-safe counterparts. Whether you're a seasoned developer or just starting out, this guide will help you navigate the world of Java collections.

Java provides a rich set of built-in collections classes that cater to different use cases. These classes are part of the `java.util` package and include:

a. Lists

- **ArrayList:** A dynamic array that grows as needed. It's fast for random access but slow for insertions and deletions.

- **LinkedList:** A doubly-linked list that's efficient for insertions and deletions but slower for random access.

Thread-Safe Alternatives:

- **CopyOnWriteArrayList:** A thread-safe version of `ArrayList`. When we perform operations like adding or modifying elements, the `CopyOnWriteArrayList` creates a fresh copy of the underlying array. While this approach can be expensive, it becomes efficient when read operations significantly outnumber mutations.
- **Vector:** Vector is an older thread-safe list implementation. It ensures thread safety by using **synchronized** methods for all operations (such as adding, removing, and accessing elements). While it provides thread safety, this synchronization can impact performance, especially in highly concurrent scenarios.
- **Collections.synchronizedList():** We can create thread safe list using Collections Utility class as well. This function returns a wrapper (`SynchronizedList`) backed on the specified list. All of the operations (`get`, `update`, `remove` etc) on the wrapper are synchronized which helps in providing the thread safety but may impact performance due to synchronization.

b. Sets

- **HashSet:** An unordered set that doesn't allow duplicate elements.
- **TreeSet:** A sorted set which maintains elements in their natural order or by a comparator passed during its creation.

Thread-Safe Alternatives:

- **ConcurrentSkipListSet:** `ConcurrentSkipListSet` is a thread-safe sorted set based on skip lists.
- **CopyOnWriteArraySet:** The `CopyOnWriteArraySet` is a thread-safe implementation of the `Set` interface, backed by a `CopyOnWriteArrayList` i.e. each update operation creates a separate cloned copy of the set.
- **Collections.synchronizedSet():** We can create thread safe set using Collections Utility class as well. This function returns a

wrapper (`SynchronizedSet`) backed on the specified set. Similar to `SynchronizedList`, in `SynchronizedSet` all of the operations are synchronized.

c. Maps

- **HashMap:** An unordered map that stores key-value pairs. It doesn't allow duplicate keys.
- **TreeMap:** A sorted map which maintains keys in their natural order or by a comparator passed during its creation.

Thread-Safe Alternatives:

- **ConcurrentHashMap:** A highly efficient thread-safe map. The map is divided into segments, and each segment is independently synchronized. It balances thread safety and performance, making it suitable for multi-threaded applications.
- **ConcurrentSkipListMap:** It is a powerful thread-safe implementation of a sorted map. It is based on skip lists, a data structure that allows efficient search, insertion, and removal operations while maintaining sorted order. Skip lists are similar to balanced trees but use probabilistic balancing rather than strict balancing rules.
- **Collections.synchronizedMap():** We can create thread safe map using Collections Utility class as well. This function returns a wrapper (`SynchronizedMap`) backed on the specified Map. Similar to `SynchronizedList`, in `SynchronizedMap` all of the operations are synchronized.

In this comprehensive guide, we've explored various Java collections and their thread-safe alternatives. However, it's important to note that we haven't covered every collection available in the Java ecosystem. There are more specialized collections, such as **BlockingQueue**, **ConcurrentLinkedQueue**, and **ConcurrentNavigableMap**, which cater to specific concurrency requirements.

Remember that choosing the right collection depends on your application's needs, performance considerations, and the level of thread safety required. Whether you're a seasoned developer or just

starting out, stay curious and explore the vast world of Java collections!

For more content and updates, follow us on our platform. Happy coding! 🚀