Java Data Structures and Thread Safety

This document explains the commonly used data structures in Java: ArrayList, LinkedList, Vector, and Stack. It also discusses their thread safety and possible alternatives for thread-safe operations.

---

1. ArrayList

An `ArrayList` is a resizable array implementation of the `List` interface.

- **Key Features**:
- Fast random access.
- Allows duplicates and null values.
- Ideal for storing and accessing data sequentially.
- **Thread Safety**: Not thread-safe.
- **Thread-Safe Alternatives**:
- `CopyOnWriteArrayList`
- `Collections.synchronizedList(new ArrayList<>())`

---

2. LinkedList

A `LinkedList` is a doubly linked list implementation of the `List` and `Deque` interfaces.

- **Key Features**:
- Efficient insertions and deletions.
- Can be used as a stack, queue, or deque.
- **Thread Safety**: Not thread-safe.
- **Thread-Safe Alternatives**:
- `Collections.synchronizedList(new LinkedList<>())`
- `ConcurrentLinkedQueue` or `ConcurrentLinkedDeque` (for queue/deque use cases)

---

3. Vector

A `Vector` is a legacy synchronized implementation of a growable array of objects.

- **Key Features**:
- Similar to `ArrayList` but synchronized.
- Every method is thread-safe by default.
- **Thread Safety**: Thread-safe (legacy synchronization).
- **Modern Alternatives**:
- `CopyOnWriteArrayList`
- `Collections.synchronizedList(new ArrayList<>())`

---

4. Stack

A `Stack` extends `Vector` and follows LIFO (Last In First Out) behavior.

- **Key Features**:
- Uses `push()`, `pop()`, and `peek()` operations.
- Inherits synchronization from `Vector`.
- **Thread Safety**: Thread-safe (legacy synchronization).
- **Modern Alternatives**:
- `Deque` (like `ArrayDeque`)
- `ConcurrentLinkedDeque`

---

5. CopyOnWriteArrayList: Internal Working and Thread Safety

`CopyOnWriteArrayList` is a thread-safe variant of `ArrayList`, ideal for read-heavy scenarios.

- **How It Works Internally**:

- It maintains a **separate copy of the underlying array** for every write operation.

- When you modify (add/remove/set), it creates a **new copy of the entire array**, makes the change, and then replaces the reference.

- Read operations use the current immutable snapshot of the array.

- **Why It's Thread-Safe**:

- Read operations do **not require locks**, as they work on an immutable snapshot.

- Write operations are synchronized, but they work on a separate copy, avoiding interference.

- This avoids the need for explicit locking during reads.

- **Use Cases**:

- Best suited when there are **frequent reads and infrequent writes**.

- Commonly used in event listener lists, caching, etc.

- **Drawbacks**:

- High memory usage due to copying.

- Poor performance for write-intensive operations.

---

6. How CopyOnWriteArrayList Prevents Write Conflicts

To prevent multiple threads from writing simultaneously and overwriting each other's changes, `CopyOnWriteArrayList` uses a **ReentrantLock** to synchronize all mutative operations.

- If **Thread A** calls `add("A")`, it acquires the lock, copies the array, modifies it, and replaces the original array reference.
- While Thread A holds the lock, **Thread B** (calling `add("B")`) must wait until Thread A completes.
- This ensures that each modification happens on the most recent version of the array and no updates are lost.

Example (Simplified):

```
final ReentrantLock lock = new ReentrantLock();

public boolean add(E e) {
    lock.lock();
    try {
        Object[] elements = getArray();
        Object[] newElements = Arrays.copyOf(elements, elements.length + 1);
        newElements[elements.length] = e;
        setArray(newElements);
        return true;
    } finally {
        lock.unlock();
    }
}
```

- **Outcome**: No two threads can overwrite each other's data. Each update is applied sequentially in a thread-safe manner.

---

7. Thread Safety Overview and Alternatives

Data Structure	Thread-Safe	Recommended Alternative (Thread-Safe)
ArrayList	❌	`CopyOnWriteArrayList`, `synchronizedList`
LinkedList	❌	`synchronizedList`, `ConcurrentLinkedDeque`
Vector	✅(legacy)	`CopyOnWriteArrayList`
Stack	✅(legacy)	`ArrayDeque`, `ConcurrentLinkedDeque`

Conclusion

While some legacy structures like `Vector` and `Stack` are thread-safe, they are not recommended for modern applications. Instead, use concurrent utilities like `CopyOnWriteArrayList` or `ConcurrentLinkedDeque` for better performance and design clarity in multi-threaded environments.