Copilot

Certainly! Here are some interview questions focused on Java Collections, specifically for experienced developers:

Advanced Questions for Experienced Developers

1. **Explain the internal working of HashMap.**
   - **Answer**: HashMap uses an array of buckets to store key-value pairs. Each bucket is a linked list or a tree (since Java 8). The position of the bucket is determined by the hash code of the key. When a collision occurs (i.e., multiple keys have the same hash code), the elements are stored in a linked list or tree within the same bucket.
2. **What are the differences between HashMap and ConcurrentHashMap?**
   - **Answer**:
     - **HashMap**: Not thread-safe, can lead to data inconsistency if accessed by multiple threads simultaneously.
     - **ConcurrentHashMap**: Thread-safe, allows concurrent access by multiple threads without locking the entire map. It uses a segmented locking mechanism to achieve this.
3. **How does the `remove` method work in a LinkedList?**
   - **Answer**: The `remove` method in LinkedList involves updating the pointers of the previous and next nodes to exclude the node being removed. If the node is at the beginning or end, the head or tail pointers are updated accordingly.
4. **What is the difference between Synchronized Collections and Concurrent Collections?**
   - **Answer**:
     - **Synchronized Collections**: Use synchronized methods to ensure thread safety, which can lead to performance bottlenecks.
     - **Concurrent Collections**: Designed for concurrent access with better performance, using techniques like lock stripping and non-blocking algorithms.
5. **Explain the concept of CopyOnWriteArrayList.**
   - **Answer**: CopyOnWriteArrayList is a thread-safe variant of ArrayList where all mutative operations (add, set, remove, etc.) are implemented by making a fresh copy of the underlying array. This is useful in scenarios where read operations vastly outnumber write operations.
6. **How does a TreeMap maintain its order?**
   - **Answer**: TreeMap maintains its order using a Red-Black tree, a self-balancing binary search tree. The keys are sorted based on their natural ordering or by a provided comparator.
7. **What are the advantages and disadvantages of using a LinkedHashMap?**
   - **Answer**:
     - **Advantages**: Maintains insertion order, can be used to create a LRU (Least Recently Used) cache.
     - **Disadvantages**: Slightly slower than HashMap due to the overhead of maintaining a linked list.

Example Code Questions

1. **How do you implement a LRU Cache using LinkedHashMap?**

    - **Example**:

    ```
    class LRUCache<K, V> extends LinkedHashMap<K, V> {
        private final int capacity;

        public LRUCache(int capacity) {
            super(capacity, 0.75f, true);
            this.capacity = capacity;
        }

        @Override
        protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
            return size() > capacity;
        }
    }

    // Usage
    LRUCache<Integer, String> cache = new LRUCache<>(3);
    cache.put(1, "one");
    cache.put(2, "two");
    cache.put(3, "three");
    cache.put(4, "four"); // Removes the eldest entry (1, "one")
    ```

2. **How do you safely iterate over a ConcurrentHashMap?**

    - **Example**:

    ```
    ConcurrentHashMap<String, String> map = new ConcurrentHashMap<>();
    map.put("key1", "value1");
    map.put("key2", "value2");

    for (Map.Entry<String, String> entry : map.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }
    ```

3. **How do you sort a list of custom objects using a Comparator?**

    - **Example**:

    ```
    class Person {
        String name;
        int age;

        Person(String name, int age) {
            this.name = name;
            this.age = age;
        }

        @Override
        public String toString() {
            return name + " (" + age + ")";
        }
    }

    List<Person> people = new ArrayList<>();
    people.add(new Person("Alice", 30));
    people.add(new Person("Bob", 25));

    people.sort(Comparator.comparingInt(person -> person.age));

    for (Person person : people) {
        System.out.println(person);
    }
    ```

These questions and examples should help you prepare for interviews focusing on Java Collections. Let me know if you need more details or have any other questions!